modern microprocessor system design

modern microprocessor system design:

SIXTEEN-BIT AND BIT-SLICE ARCHITECTURE

Daniel R. McGlynn

001.640445
MAC
162.455
Mar.81
Quarto

A WILEY-INTERSCIENCE PUBLICATION

JOHN WILEY & SONS
New York • Chichester • Brisbane • Toronto

Copyright © 1980 by John Wiley & Sons, Inc.

All rights reserved. Published simultaneously in Canada.

Reproduction or translation of any part of this work beyond that permitted by Sections 107 or 108 of the 1976 United States Copyright Act without the permission of the copyright owner is unlawful. Requests for permission or further information should be addressed to the Permissions Department, John Wiley & Sons, Inc.

Library of Congress Cataloging in Publication Data:

McGlynn, Daniel R
Modern microprocessor system design.

"A Wiley-Interscience publication."
Includes index.
1. Microprocessors. 2. Minicomputers.

TK7895.M5M32 001.64'04 80-12549
ISBN 0-471-06492-0

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

preface

Significant advances have been made in microprocessor technology since the publication of our book *Microprocessors: Technology, Architecture, and Applications* in 1976. One of the most important of such advances has been the development and application of sophisticated 16-bit and bit-slice microprocessors. *Modern Microprocessor System Design* presents a review and survey of such microprocessors, along with recent developments in interfacing, system development, and standardization of signal buses.

In the last few years, microprocessor technology has evolved in two prominent directions:

- The design of microprocessors that have more and more the features of large-scale computers.
- The design of microprocessors more closely adapted to the digital hardware or application.

The first direction may be characterized by the development of 16-bit microprocessors such as the Intel 8086, the Zilog Z8000, and the Motorola 68000. The second direction may be characterized by the development and widespread use of bit-slice microprocessors, such as the Advanced Micro Devices 2901.

The 16-bit and bit-slice microprocessors are far more complex than the well-known general-purpose 8-bit microprocessors. Nevertheless, the electronic designer and computer system analyst today must have a basic familiarity with such devices, whether or not they plan to use such devices in the products they design or those with which they work. This book provides an introduction to this new technology.

Chapter 1 is an overview of the microcomputer industry and places the 16-bit microprocessor in perspective, both in terms of evolutionary development and application. Since the economics of microcomputer technology is of considerable interest, a large portion of the chapter is devoted to price and performance characteristics and future trends. The recent concern in the U.S. semiconductor industry with the impact of international technology is also reviewed.

Chapter 2 is a survey of the architecture of a variety of 16-bit microprocessors. Chapters 3 and 4 examine the hardware and instruction set of one of these—the Intel 8086—in detail, while Chapter 5 examines the Zilog Z8000.

Chapter 6 turns to the bit-slice microprocessors and surveys the architecture of three popular devices. Chapter 7 examines one of these—the Advanced Micro Devices 2901 —in greater detail.

Recent developments in peripheral devices and interfacing to such devices is considered in Chapter 8. Analog interfacing is covered in Chapter 9, while Chapter 10 examines the various bus standards popular in microcomputer systems.

Chapter 11 considers the overall task of system development and surveys the various tools used to develop and test a prototype microcomputer system. Chapter 12 reviews the subject of hardware reliability from the perspective of both the device and the system.

Daniel R. McGlynn

Anaheim, California
February 1980

contents

1 THE MICROCOMPUTER INDUSTRY, 1

The Microprocessor Market, 1

Technological Evolution, 5

Price and Performance Characteristics, 6
Future Technological Trends, 10

Lithography and Semiconductor Process Technology, 11

Improvements in Materials, 13

Circuit Design, 14

The Semiconductor Industry, 14
The Impact of International Technology, 17

International Trade Aspects, 17

International Economic Issues, 19

Technological Leadership, 21

The Japanese Challenge, 27

Soviet Developments, 31

2 THE SIXTEEN-BIT MICROPROCESSORS, 34

Intel 8086, 34
Zilog Z8000, 35
Texas Instruments TMS9900, 37
Texas Instruments TMS9940, 44
Fairchild 9440 Microflame, 51
Motorola 68000, 57
Ferranti F100-L, 63
Data General mN602, 63
National 16000 Family, 64
Advanced Micro Devices 29116, 64
Eight-Bit Microprocessors with 16-Bit Internal Architecture, 67

Intel 8088, 67

Future 32-Bit Microprocessors, 68

3 INTEL 8086 SYSTEM, 74

Internal Architecture, 74
System Configuration, 76
System Timing, 79
Instruction Set, 82
memory addressing, 85

4 INTEL 8086 PROGRAM DEVELOPMENT, 115

Assembly Language Programming, 115
Effective Address Computation, 116

mod = 00, 119

mod = 01, 120

mod = 10, 120

Use of Segment Overide Prefix, 122

The Detailed 8086 Instruction Set, 122

5 ZILOG Z8000 SYSTEM DESIGN, 127

Internal Architecture, 127
System Configuration, 130
System Timing, 133

Memory Read and Write, 133

Interrupt and Segment Trap Request/Acknowledge, 133

Instruction Set, 134

6 BIT-SLICE MICROPROCESSORS, 148

Microprogramming, 148
AMD 2901/2901A, 149
AMD 2903, 154
Motorola 10800, 158
Texas Instruments SN 54S/74S-481, 162
Fairchild F100220, 162

7 AMD 2901 SYSTEM DESIGN, 167

Microprogrammed Design, 167
System Configuration, 168
A Basic Computer Controller, 172
Instruction Set, 176
Software, 176

8 INTERFACING, 182

Memory Interfacing, 183
RAMs, 184
Mass Memories, 192
I/O Interface Control, 209
The 8255 Programmable Peripheral Interface, 211
CRT Interfacing, 213
Character Generation, 214
Video Interface Circuits, 214
Serial Communications Device Interfacing, 220
Multiprotocol Data Link Control Interfacing, 223
Encrypted Data Link Interfacing, 225
Floating Point Processors, 225
AMD 9511, 225

9 ANALOG INTERFACING, 230

Sensors, 231
Data Conversion Devices, 233
A/D Converters, 233
D/A Converters, 234
Implementations, 235
Analog Signal Processing, 235
Intel 2920, 235

10 BUS STANDARDS, 237

Multibus, 237
LSI-11 Bus, 238

11 SYSTEM DEVELOPMENT, 241

Evaluation Systems, 241
MCS-86 System Design Kit (SDK-86), 242
Microprocessor Development Systems, 244
Single Board Computers, 245
Logic Analyzers, 246
Logic State Analyzers, 246
Logic Timing Analyzers, 251
Field Test Equipment, 253

12 RELIABILITY OF MICROCOMPUTER SYSTEMS, 255

Device Reliability, 255
Failure Mechanisms, 255
Reliability Testing, 257
Quality Assurance, 259
Operation in Severe Environments, 261
System Reliability, 263
System Quality Assurance, 264
Self-Diagnosis and Correction, 264
External Error Detection/Correction, 264
Redundant Systems, 265

APPENDIX 1 MICROPROCESSORS AND MICROPROCESSOR MANUFACTURERS, 267

APPENDIX 2 SINGLE BOARD MICROCOMPUTER SYSTEMS, 271

BIBLIOGRAPHY, 283

GLOSSARY, 285

INDEX, 293

1

the microcomputer industry

Since the introduction of the microprocessor in the early 1970s, the microcomputer industry has grown to be one of the most important technological driving forces in the electronics and data processing industry today. Before embarking on a detailed technical examination of some specific products and techniques in the industry, it would be worthwhile to put the microcomputer industry in perspective with respect to the electronic and semiconductor industries, and to consider the microprocessor from a market perspective. Because of the importance of high-technology products in economic growth, it would also be useful to note the "Japanese challenge" and its effects on the semiconductor industry and in consumer electronics.

In this chapter, we direct our attention to:

- The microprocessor market.
- Price and performance characteristics.
- Future technological trends.
- The semiconductor industry.
- The impact of international technology.

The Microprocessor Market

In our book *Microprocessors: Technology, Architecture, and Applications* (Wiley, 1976), we suggested that one of the important trends that would shape the future microcomputer industry was increasing product differentiation and market segmentation. In examining the microprocessor market today, we can clearly see such a trend emerging.

The microprocessor market can presently be viewed as being segmented into three distinct areas:

- A 4-bit or low-end 8-bit "low-cost controller" market.
- An 8-bit midrange "byte handling" market.
- A 16-bit "computation" market.

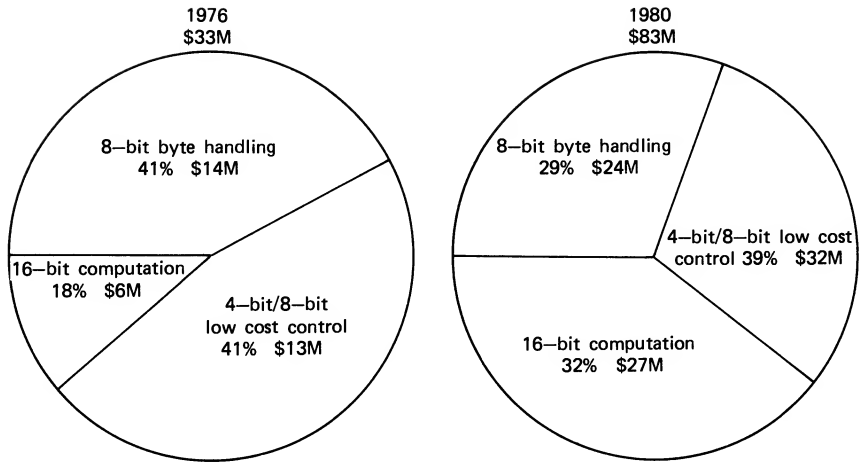


FIGURE 1.1 World market for microprocessors.

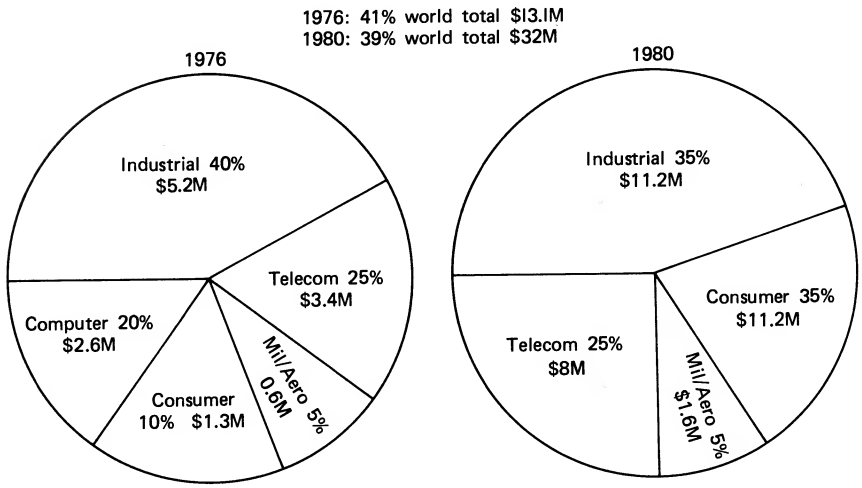


FIGURE 1.2 The 4-bit and 8-bit low cost control market.

Where 4-, 8-, and 16-bit refer to the word size of the microprocessor.

A 1977 market research study by a major semiconductor manufacturer indicated how these three market segments make up the overall microprocessor market, based on 1976 actual and 1980 projected data. The results of the study are presented in Figure 1.1.

Market segmentation is basically by application. The key market application areas for microprocessors are:

- Industrial.
- Computer.
- Telecommunications.
- Consumer.
- Military/aerospace.

Each of these markets has requirements for each of the three basic types of microprocessors in different proportions. The research study noted above therefore considers the breakdown by application of each of the three basic types of microprocessors in Figure 1.2, 1.3, and 1.4.

Considering just one market application area—consumer products—we can examine the breakdown into specific products of microprocessor usage, as shown in Table 1.1.

A more detailed breakdown of microprocessors gives five distinct types:

- Very low end.
- Low end.
- Midrange standard.
- Midrange enhanced.
- Upper end.

TABLE 1.1. Microprocessor Applications in Consumer Products (Millions of Units, Worldwide)

Year	Toys and Games	Calculators	Microwave Ovens	Automobiles	Personal Computers
1976	6.0	1.6	0.7	—	0.05
1978	8.0	5.0	3.0	1.0	0.2
1980 (est.)	17.0	10.0	6.0	3.0	0.6
1985 (est.)	50.0	25.0	10.0	10.0	1.5

The very low-end microprocessors are 4-bit microcomputers on a chip. Examples include the TI TMS 1000 and 1100, the Rockwell MM76/77, and the National Semiconductor 5734 and 5799. Such microprocessors are directed to the simplest consumer and industrial control applications.

The low-end microprocessors are the simple 8-bit microcomputers on a chip. These include the Mostek 3870, the Intel 8048/8748, and the Zilog Z-8. Such microprocessors are again directed to simple consumer and industrial control applications, but these applications require more sophisticated processing capability.

The midrange-standard microprocessors are the majority of the general-purpose microprocessors on the market today. Examples include the Intel 8080, the Motorola 6800, and the MOS Technology 6502. These microprocessors are directed to the bulk of the industrial, computer, and telecommunications applications of microprocessors today.

The midrange-enhanced microprocessors are those 8-bit microprocessors that feature the latest technology and greatest capabilities. Examples include the Zilog Z80 and Intel 8085. These microprocessors are grad-

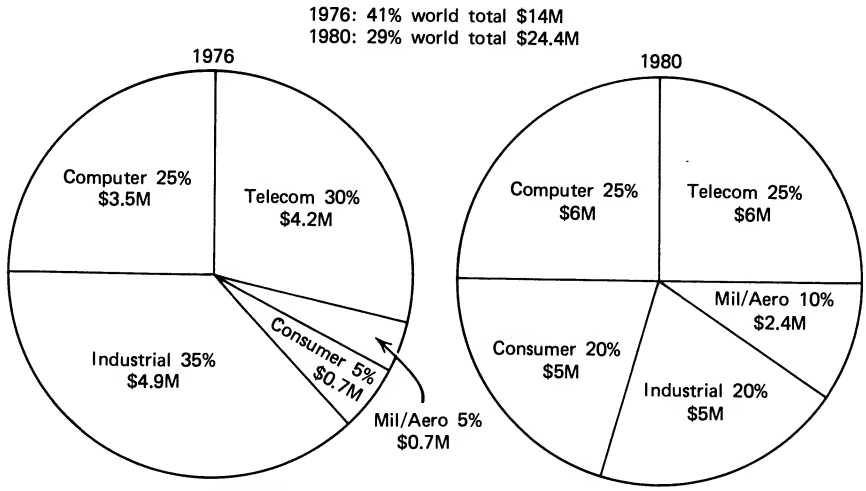


FIGURE 1.3 The 8-bit byte-handling market.

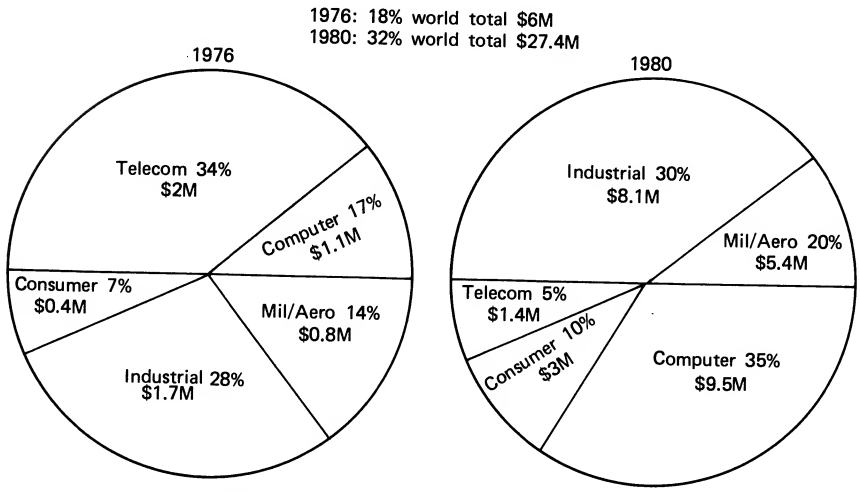


FIGURE 1.4 The 16-bit computation market.

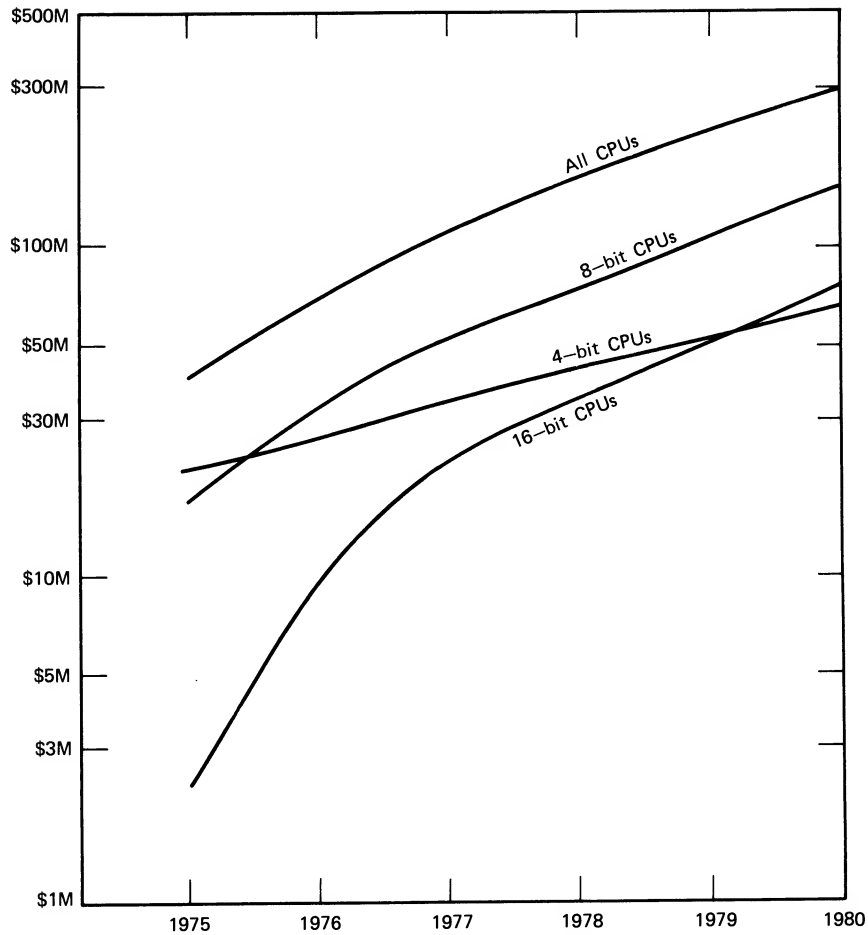


FIGURE 1.5 The market for CPUs 1975-1980.

ually replacing many of the midrange-standard microprocessors in applications where additional speed or processing capability is required.

The upper-end microprocessors are those 16-bit microprocessors that are designed for the most sophisticated applications in the market. Examples include the TI TMS 9900, the Intel 8086, and the Zilog Z8000. This 16-bit "computation" market is quite distinct from the 8-bit market and is the most rapidly growing.

Another category of microprocessors is "bit-slice" microprocessors. These products are implemented in microprogrammed systems and thus would properly fall within the "computation" market that we consider above, even though such systems need not necessarily be 16 bits.

The total market for microprocessors—4-, 8-, and 16-bit varieties—is summarized in the graph in Figure 1.5. It is projected that the total worldwide market for CPUs in 1980 will amount to $283 million. The 8-bit CPUs will still be the dominant market product, accounting for 50% of the total market. The 16-bit CPUs will account for some 26% of the market in value, slightly more than the 24% projected for the 4-bit CPUs. The most important point to note from this graph is the rapid increase in importance of the 16-bit CPU in the 1978–1980 time period, occasioned by the introduction of new 16-bit microprocessors by the major manufacturers during that time period.

The microprocessor market alone should not be considered when examining the microcomputer industry. As we pointed out in our earlier book, the memory and I/O components constitute an appreciable portion of the total system cost. As semiconductor technology advances, the microprocessor component cost for a given capability decreases, and thus the relative importance of the microprocessor as a percentage of total system cost is expected to decrease significantly as well. Figure 1.6 quantifies this conclusion by considering microprocessors and the "drag business" (memories and I/O devices) over the period 1975–1980. It is noted that the most significant component of system cost was the microprocessor in 1975, but by 1980 it is expected that memory will account for 60% of the market, the CPU only 25%, and I/O 15%. In other words, the most important component of system cost in 1980 will be memory, reflecting perhaps not only increasing memory

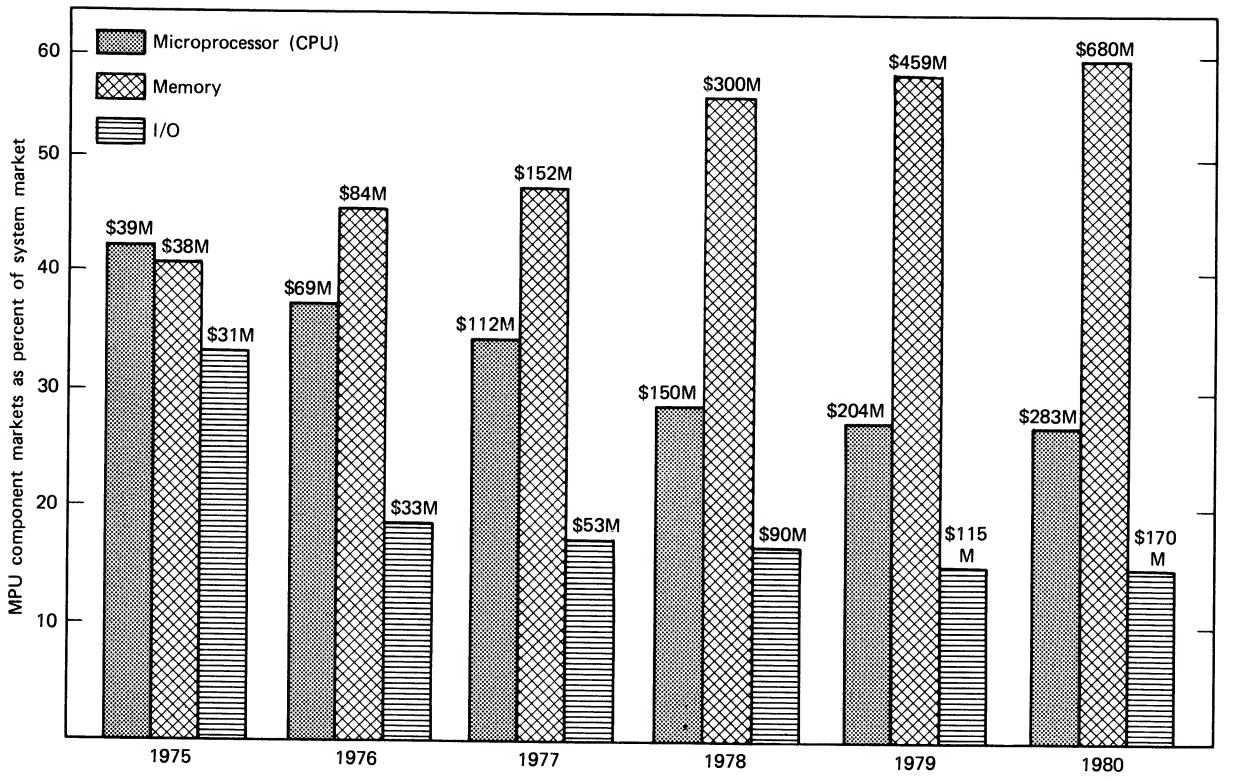


FIGURE 1.6 Microprocessors and drag business (world markets) 1975–1980.

utilization with more sophisticated software systems, but also the decreasing cost of the microprocessor.

The total world market for microprocessor systems (i.e., complete microcomputers) is summarized in the graph in Figure 1.7. It is projected that by 1980 the total world market for systems will be $1.13 billion, with the United States and Canada accounting for 54%, Japan for 23%, Europe for 14%, and the rest of the world for 9%.

A more recent market study by Creative Strategies of San Jose, California predicted similar values, although it indicated the penetration of the 16-bit computation oriented microprocessors was slower and less extensive than originally suggested back in 1977. The Creative Strategies study suggested that the worldwide market for microprocessor systems will reach $1.3 billion by 1983, with the 8-bit units retaining over 60% of the market share. In the 5-year period 1978–1983, the market share of 16-bit units is expected to rise from 6 to 23%.

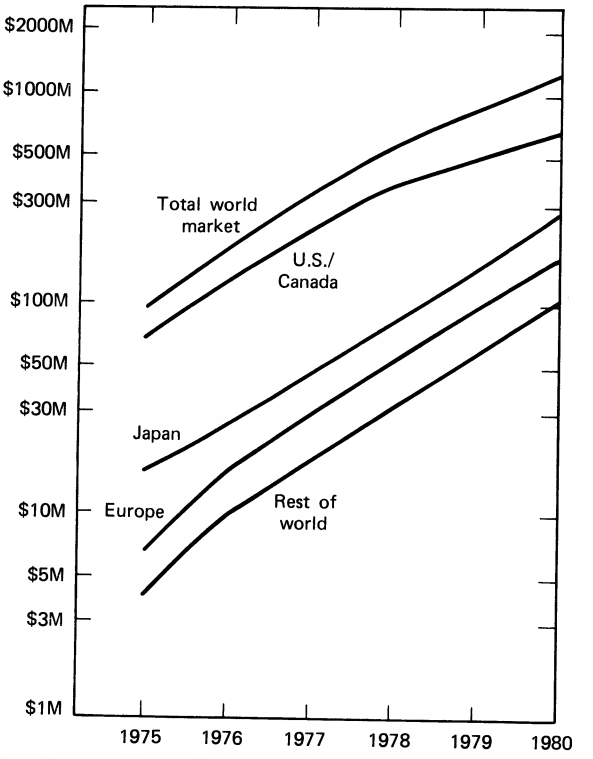


FIGURE 1.7 The world market for microprocessor systems 1975–1980.

Technological Evolution

In examining and comparing new microprocessors and related components, the issue of technological evolution must be kept in mind. Technological evolution is a term used to describe the introduction of new products that improve upon older products. As an example, con-

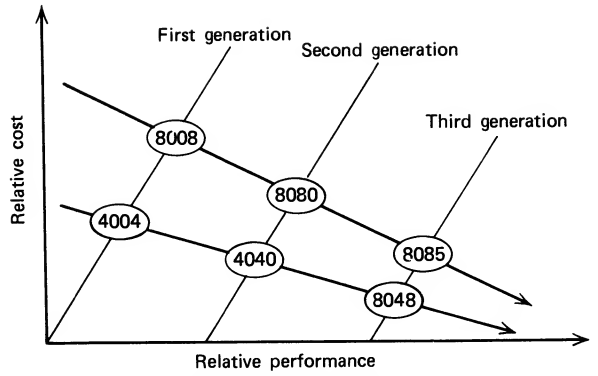


FIGURE 1.8 Technological evolution: the Intel family.

sider the "evolution" of microprocessors within the Intel product offering, as represented in Figure 1.8.

Figure 1.8 shows the evolution of two distinct families of processors: the 4-bit or low-end 8-bit control processors (the 4004, 4040, 8048) and the general-purpose 8-bit processors (8008, 8080, 8085). Note that with each new product introduced (i.e., the 4040 following the 4004, the 8048 following the 4040, etc.), a significant improvement in both price and performance is achieved. A corollary of such technological evolution is the technological "obsolescence" of the previous generation of processors.

To avoid the negative consequences of technological obsolescence, Intel and other major manufacturers have designed later generations in a family of microprocessors to be relatively similar to the older generation, thus making the transition from products designed with the older generation to products designed with the new generation easier. The manufacturers do this by making the software of the new generation products compatible at some level with the software of the previous generation, and even keeping similar hardware or register structures so as to simplify the conversion task. An example of such generational compatibility is demonstrated later in our comparison of the Intel 16-bit 8086 with the older 8080. This evolutionary design of new processors is not something unique with the microprocessor industry. The same strategy has long been used by mainframe manufacturers such as IBM, which employed this strategy in the evolution from System/360 to System/370 and later the 303X series, so that customers who developed software on previous models may be able to upgrade their hardware to the new systems without significant software changes. Such a strategy is clearly necessary if a manufacturer expects to retain its customer base in view of the proliferation of new hardware manufacturers that offer equal or superior price/performance characteristics.

It must be emphasized that the technological evolution shown in Figure 1.8 does not imply that the 8080 "replaced" the 8008 or that the 8085 will "replace" the 8080. As we pointed out in *Microprocessors: Technology, Architecture, and Applications,* the 8080 was *not* a replacement for the 8008. It was another step in the technological evolution of a family of compatible microprocessors. Similarly, the 8085 did not replace the 8080. Intel, and other manufacturers of the 8080, will continue to produce that product as long as customer demand warrants such production. If one wishes to talk of "replacement," one might properly say that the Intel 8080A replaced the Intel 8080, but such a discussion is more one of semantics rather than technological strategy.

Listed below are the distinct milestones in the microprocessor life cycle:

- Introduction.
- Distributor stocking.
- Initial design-in.
- Design-in prototypes move into production.
- Second sourcing.
- Commodity stage.
- Obsolescence.

At the time of this writing (late 1979), the 8086 is at the design-in stage, while the 8080 is at the commodity stage.

Price and Performance Characteristics

It is important to develop some objective price and performance characteristics in comparing microprocessors. In this section, we first examine a typical microprocessor "figure of merit." In developing such a parameter, it must be emphasized that its purpose is not to compare the capabilities of a given processor for a specific application, but to give some general measurement of processor capability or performance. One such figure of merit is:

$$\text{figure of merit} = \frac{\text{word length} \times \text{number of instructions}}{\text{chip area} \times \text{power dissipation} \times \text{instruction cycle time}}$$

Increased performance of a microprocessor is associated with longer word lengths or a greater number of instructions. The parameters of word length and number of instructions are both quantities that are directly proportional to the figure of merit as defined above. The power dissipation (which is related to the number of gates on the chip) and the instruction cycle time (which is related to the architecture and timing characteristics

of the processor) are both quantities that decrease with increasing performance or figure of merit. Thus these two pairs of quantities are on opposite sides of the division symbol. Finally, processor capability is related to the size of the processor, or more accurately, the size of the chip (clearly a larger processor has more capability than a small processor). Thus the chip area is inserted in the figure of merit as a normalizing factor.

It must be emphasized that the above defined figure of merit parameter is only an example. The choice of word length, power dissipation, or other parameters as defining a figure of merit is merely to indicate some typical physical characteristics associated with microprocessors. Whether one or more such characteristics should have more weight than another in defining the "merit" of a microprocessor is an application dependent question and cannot be answered categorically.

Another disclaimer also must be made, that is, that the value of such parameters may vary widely even within a given microprocessor type. Thus the instruction cycle time of a microprocessor depends on the clock rate, and there is a wide range of clock rate capabilities even for one microprocessor such as the 8080.

With these disclaimers in mind, we illustrate the application of the figure of merit to some well-known microprocessors in Figure 1.9. Our purpose here is to show the relative change in figure of merit from one generation of processor to another. The 8008, introduced in 1971, may be considered a "first generation" microprocessor; the 8080, introduced in 1974, a "second generation" microprocessor; the Z-80, introduced in 1976, a "third generation" microprocessor; and the 8086, introduced in 1978, a "fourth generation" microprocessor. The figures of merit of these microprocessors have changed by a factor of 10 almost every generation.

The other important factor in considering the price/performance of microprocessors is the price. There are two approaches to examining microprocessor cost:

- Cost per gate for the most advanced semiconductor circuit.
- Price of a single general-purpose microprocessor.

Figure 1.10 shows the change in the cost per gate with different generations of semiconductor technology: medium-scale integration TTL, the first generation 4004 microprocessor, and the second and third generation microprocessors 8080 and Z-80. By "gate" we mean a logic gate, such as an inverter or a NAND gate. In MOS technology, there are typically two MOS transistors per gate. To give some indication where the fourth generation microprocessors would fall in the cost/gate graph of Figure 1.10, it is noted that the Intel 8086 using the

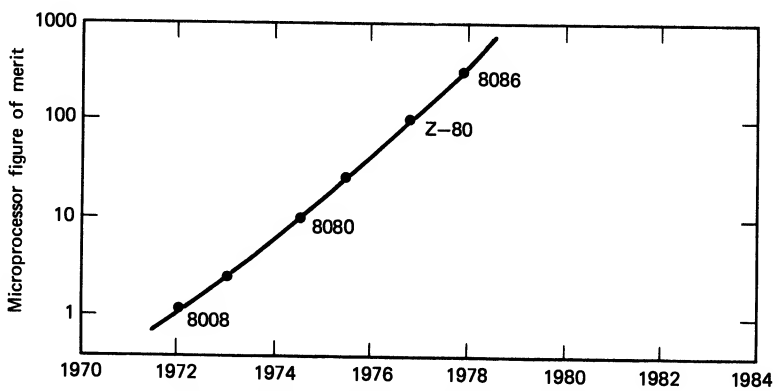


FIGURE 1.9 Microprocessor figure of merit.

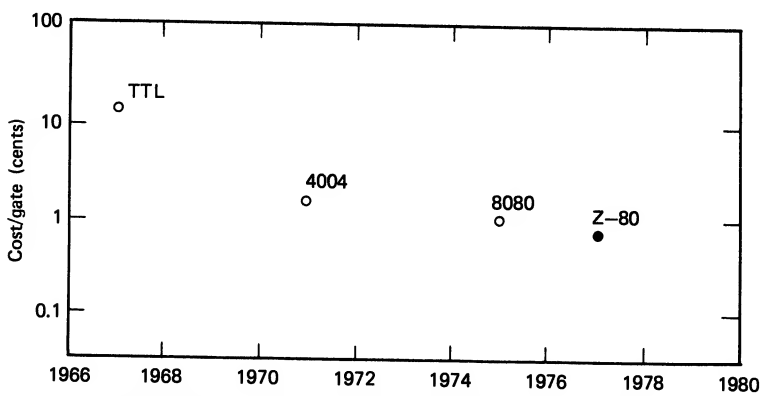


FIGURE 1.10 Cost per gate in semiconductor technology.

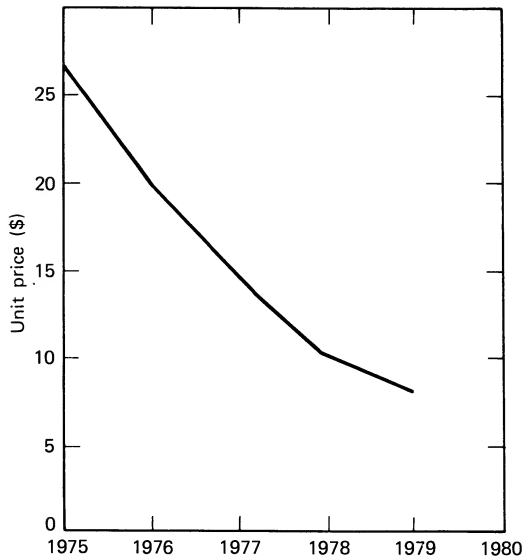


FIGURE 1.11 Microprocessor pricing trends: the 8080.

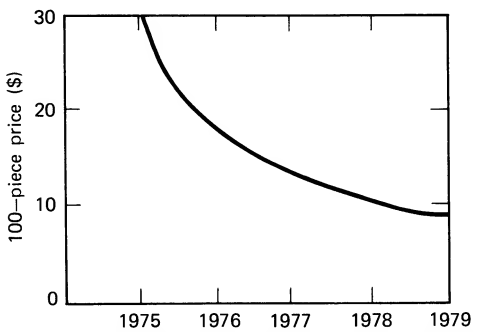


FIGURE 1.12 Microprocessor pricing trends: the 2901.

HMOS process has about 29,000 transistors, or almost 15,000 gates. At the initial marketing price at about $100.00/unit, the cost/gate is about 0.6 cents, consistent with the trend line shown. Of course as the 8086 goes into full production, the marketing price may be expected to drop substantially.

The last remark directs our attention to the other approach to examining microprocessor cost: the price of a single general-purpose microprocessor. It is evident that the initial price of any new advanced technological component does not reflect its average selling price during product maturity and full production in a competitive environment. The 8080, which is a mature microprocessor product available from a large number of different manufacturers and vendors, is a good example to choose to examine the change in unit price over time. Figure 1.11 shows 8080 pricing since its introduction in 1974. Again the downward trend in price is notable.

The pricing trend for the 8080 is not unique. Figure 1.12 shows the pricing trend for the Advanced Micro Devices (AMD) 2901 bit slice microprocessor. This particular microprocessor is discussed in more detail in later chapters.

An important factor that must be considered in analyzing pricing trends is the die size of the integrated circuit. Since the die size is closely related to production cost, the pricing trend of a product must be related to the change in die size of that product as advances in semiconductor process technology permit the same circuit to be implemented on smaller and smaller dies. Figure 1.13 shows the change in die size of the 2901 microprocessor, indicating over a 50% decrease in die size over a period of 3 years. Thus the pricing trends shown in Figures 1.11 and 1.12 not only reflect increasing competition for mature microprocessor products, but also significant decreases in production cost and improved technology. The subject of die size is further considered in the context of future trends in microprocessor design later in this chapter.

Another factor that must be considered is the improved performance of specific products as technology advances. Figure 1.14 considers the benchmark of 16-bit register-to-register add time using the 2901 microprocessor (in slices to form a 16-bit computer). The improved versions of the original 2901, designated the 2901A and 2901B, are shown in the figure and indicate the trend in improved performance.

As we point out earlier, the microprocessor is only one component of a microcomputer system. Memory is also an important factor, and a similar figure of merit for memory can also be developed. Again, it must be

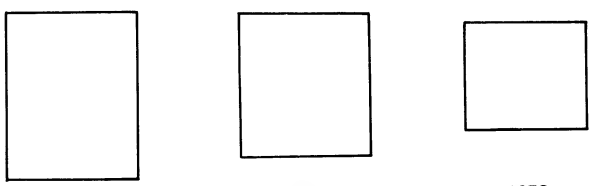


FIGURE 1.13 Microprocessor die size trends: the 2901.

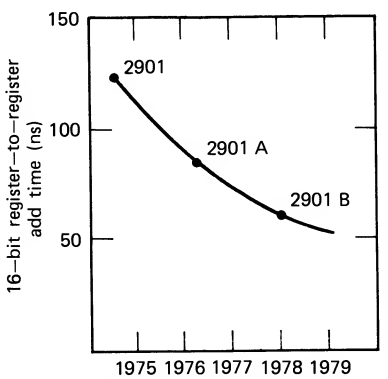


FIGURE 1.14 Microprocessor performance trends: the 2901.

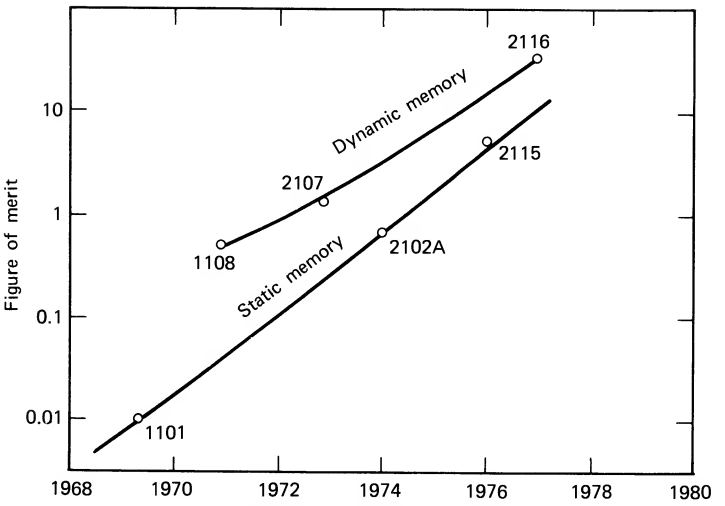


FIGURE 1.15 RAM figure of merit.

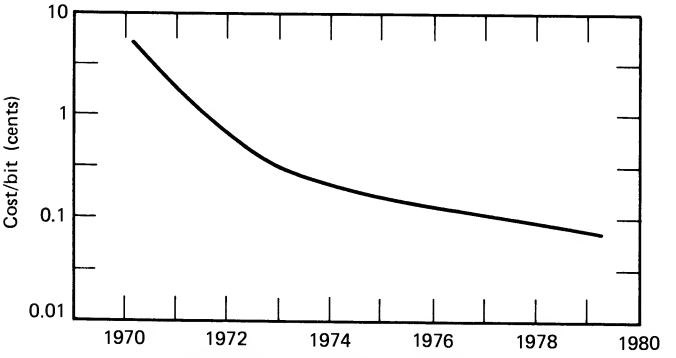


FIGURE 1.16 RAM cost per bit.

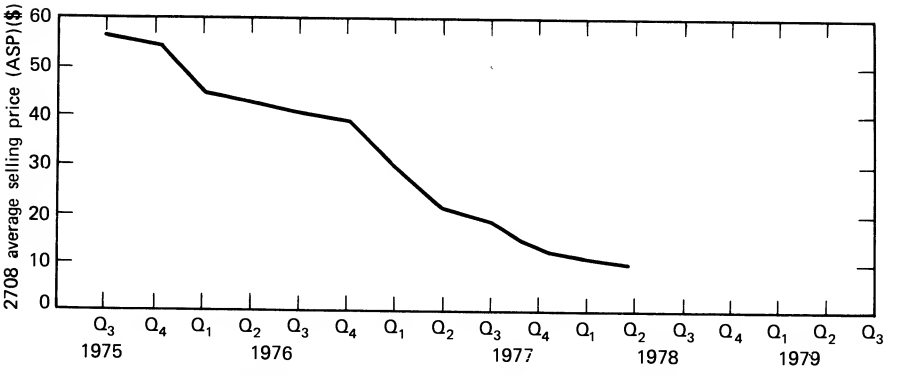


FIGURE 1.17 E-PROM pricing trends: the 2708.

pointed out that the purpose of developing such a figure of merit is not to create some absolute measurement of memory performance, but merely to form some comparative factor for evaluating the magnitude of technological change. The figure of merit for memory is:

$$\text{figure of merit} = \frac{\text{bits}}{\text{chip area} \times \text{power dissipation} \times \text{access time}}$$

An increase in the number of bits that the memory stores indicates an improved figure of merit. The power dissipation and the access time are both quantities that decrease with an improved figure of merit. The chip area is again used as a normalizing factor.

With this figure of merit for random access memory defined, we can proceed by illustrating the application of this figure of merit to some well-known random access memory (RAM) chips in Figure 1.15. Intel Corporation static and dynamic RAMs are used as examples.

As is seen in the figure, each new generation of RAM (from 1K, to 4K, to 16K bits per chip) experiences a factor of 10 change in the figure of merit over the prior generation.

The other important parameter for RAM is the cost per bit. Figure 1.16 shows the declining cost per bit of RAM since the first introduction of semiconductor memory. The downward trend in price is notable, but it must be pointed out that the decrease is not as great as that of the microprocessor. The declining price trend associated with memory is not just reflected in random access memory (RAM). Figure 1.17 shows the average selling price (ASP) of electrically reprogrammable read-only memory, the 2708.

Future Technological Trends

In examining future trends in microprocessors, it is important to note that the maximum and economic die size (i.e., the size of the "chip") is gradually increasing with time and improvements in semiconductor process technology. The die size is important since the larger the die size, the more gates and therefore the more functions that can be put into a single package. Figure 1.18 shows the maximum and most economic die size as a function of the wafer diameter.

What is the consequence of such an increase in the die size? If we suggest that the trend is toward building a single-chip microcomputer, with CPU, PROM, RAM, A/D, and I/O all in one package, then such a component is expected to be economically manufacturable by 1980–1981 (Figure 1.19). By 1981, a die size of 300 X 300 mils will be economically manufacturable, and the single-chip microcomputer would at least be an economically feasible product.

It is interesting to note that the die size of the Intel 8086 is 225 X 225 mils, the die size of the Zilog Z8000 is 238 X 256 mils, and the die size of the Motorola 68000 is expected to be 265 X 265 mils. Of the three chips, the Intel 8086 was the first to be introduced and the Motorola 68000 was the last. Evidently in terms of actual microprocessors we are slowly approaching the 300 X 300 mil size.

Of course, just as improved technology permits larger and larger die sizes, the same advancing technology permits a given integrated circuit to be implemented on smaller and smaller dies. Figure 1.13 shows the actual trend in die size for a popular microprocessor, the 2901. Thus the improvements in semiconductor technology permit not only more and more functions to be implemented in a given chip (through larger die sizes), but also the implementation of a given processing capability (e.g., represented by the 8080 or 2901 microprocessor) can be

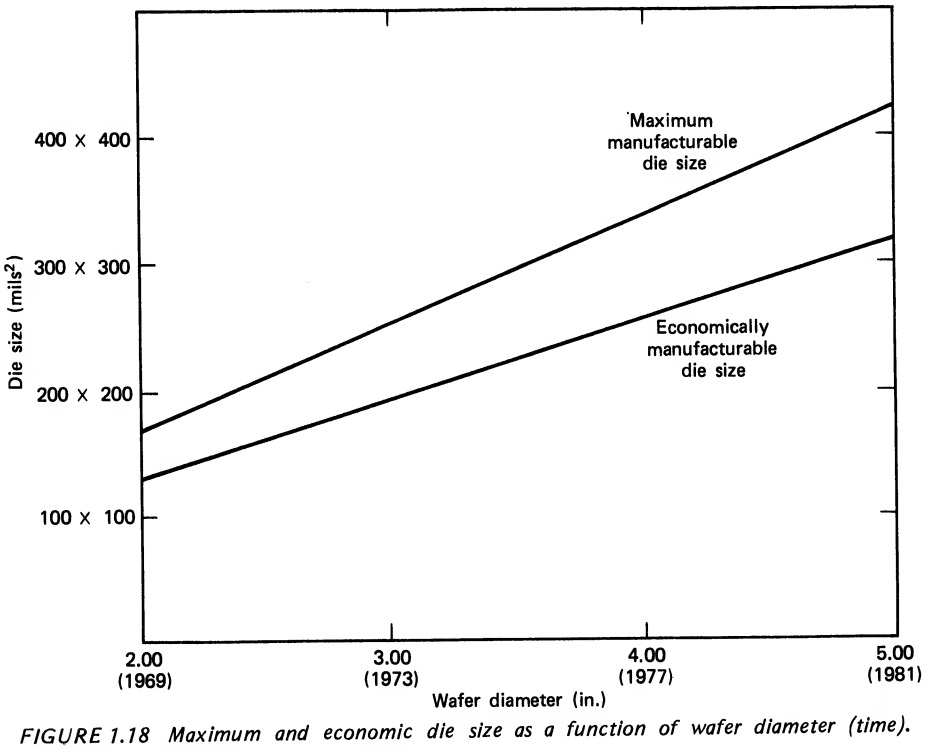


FIGURE 1.18 Maximum and economic die size as a function of wafer diameter (time).

Function	Approx. Die Size (mils)	Total Die Area ($mils^2$)
CPU (8 bit)	200 X 200	40,000
PROM 16K bit	200 X 200	40,000
RAM 1 K bit	120 X 120	14,000
8-bit A/D	130 X 130	17,000
I/O, registers, etc.	100 X 100	10,000
8-channel analog mux	70 X 70	5,000
		126,000
Less 30% efficiency obtainable by single chip layout (no pads, scribeline output stages etc. between functions)		-37,800
Single-chip die size		88,200
		≅ 300 X 300 mil
Will become economically manufacturable by early 1980's		

FIGURE 1.19. Die size estimate for single-chip super microcomputer of the 1980s.

achieved in smaller and smaller dies, and therefore at lower cost.

In addition to die size, another important consideration is the yield. The yield is the ratio of the number of "good" integrated circuits to the total number produced on the wafer. Yield is of course directly related to the die size: the smaller the die size, the more chips there are on a wafer of a given size, and the higher the yield. There are different yields for different manufacturing steps, as suggested by Table 1.2, with the cumulative yield being the key parameter. The impact of the yield on the ultimate cost of the chip is also shown in Table 1.2.

As we note earlier, the basic trend in microprocessor design is increasing product differentiation and product segmentation. The trend is not just toward "bigger and better" or "smaller and cheaper," but toward a whole range of product capabilities that attempt to satisfy the perceived industry demand for a wide range of processing capabilities.

In considering future technological trends, attention must be paid to three basic areas:

- Improvements in lithography and semiconductor process technology.
- Improvements in materials.
- Improvements in circuit design.

Lithography and Semiconductor Process Technology

There are many different types of lithography applicable to semiconductor processes:

- Contact.
- Proximity.
- Projection.
- Step and repeat.
- X-Ray.
- Electron beam.

One of the sources of such improvement in semiconductor technology is the decrease in line width made possible by improvements in lithography. Figure 1.20 shows the minimum line width, in microns, for production LSI. Table 1.3 gives the comparison between the various possible photomasking techniques.

TABLE 1.3. Photomasking Techniques

Technique	Typical Line Width (μm)
Contact printing	3-5
Projection printing	1-3
Step and repeat projection	1-3
Electron beam	0.5-1.0
X-Ray	0.1-0.5

TABLE 1.2. Estimated Manufacturing Yields for Two Different Integrated Circuits, 1975.

	T^2L[a]			1103[b]		
Manufacturing Step	Percent Yield	Cumulative Percent Yield	Cumulative Cost per Chip ($)[c]	Percent Yield	Cumulative Percent Yield	Cumulative Cost per Chip ($)[c]
Wafer fabrication	80	80	0.010	70	70	0.153
Wafer electrical test	40	32	0.025	20	14	0.926
Chip bonding and wiring	90	29	0.029	85	11.9	1.09
Final electrical test	90	26	0.033	75	8.9	1.45

SOURCE: William F. Finan, "The International Transfer of Semiconductor Technology through U.S. Based Firms," (unpublished National Bureau of Economic Research Working Paper, October 1975), tables 3-3 and 3-4, pp. 22 and 23.

[a] *Transistor-transistor-logic, a mature integrated circuit device. Assumes 2385 chips/wafer.*

[b] *Random access memory device. Assumes 140 chips/wafer.*

[c] *Cumulative costs include materials, direct labor, and overhead for each successive step; with assembly performed in Singapore.*

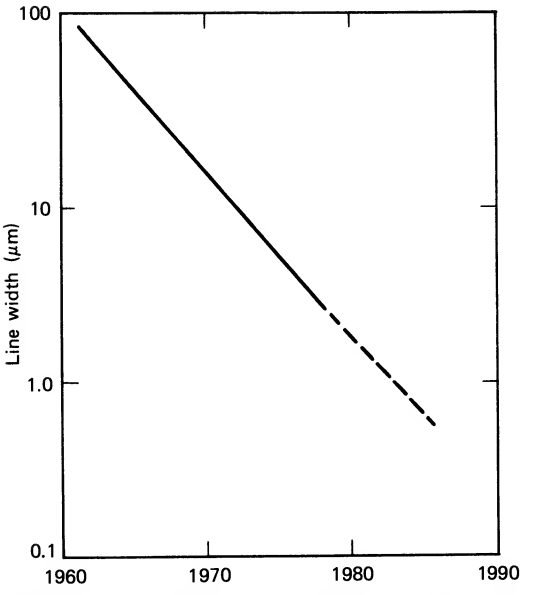


FIGURE 1.20 Minimum line width for production LSI.

The figure of merit for photomasking systems presented in Table 1.3 is defined as:

$$\text{figure of merit} = \frac{\text{throughput}}{(1 + 0.15\ \text{defects})^8 \times \text{cost} \times (\text{line width})^2}$$

This figure of merit attempts to represent the overall cost-effectiveness of a given process.

As the line width decreases and the economic die size increases, the number of electronic components or active element groups that can be implemented on a single chip increases. Figure 1.21 shows the trend with limits imposed by optical, electron beam, and X-ray lithographic methods.

In addition to lithographic improvements, there have also been improvements in techniques for high-resolution etching of the fine line patterns defined by the lithography. These techniques are generally known as dry etching, as opposed to the typical "wet" chemical etching processes.

Dry etching consists of mechanically or chemically removing material from the surface of a body to be etched. Mechanical techniques include sputter etching and ion milling, while chemical techniques include reactive ion etching and plasma etching. It would be useful to define each of these techniques more specifically at this point.

Sputter etching is the removal of material by various nonreactive heavy atoms that bombard the body and displace material therefrom.

Ion milling is similar to sputter etching except that an accelerated beam of ions is used.

Plasma etching is a process in which a partial pressure of a gas is excited creating a reactive gas plasma. The plasma then chemically reacts with the surface of the body to be etched. The advantage of plasma etching over mechanical techniques is that different gases having different etch rates can be specifically selected as required for the process.

Reactive ion etching is a variation of plasma etching in which the material to be etched is grounded.

In addition to the purely mechanical or purely chemical etching techniques, there are also process techniques that are a combination of the two. Reactive ion beam etching (RIBE) is one such example, combining reactive ion etching and ion milling.

In addition to dry processing techniques, significant advances in other processing techniques, such as laser annealing, lift-off, and plasma deposition, have also taken place.

Laser annealing refers to the use of a pulsed laser to heal radiation damage to semiconductor crystals that occurs during implantation of impurities in the semiconductor. The laser melts the material in the path of the beam, and the impurities in the semiconductor then diffuse in the liquid phase. Such diffusion can be controlled for the purpose of minimizing redistribution or to achieve a controlled movement of implanted impuri-

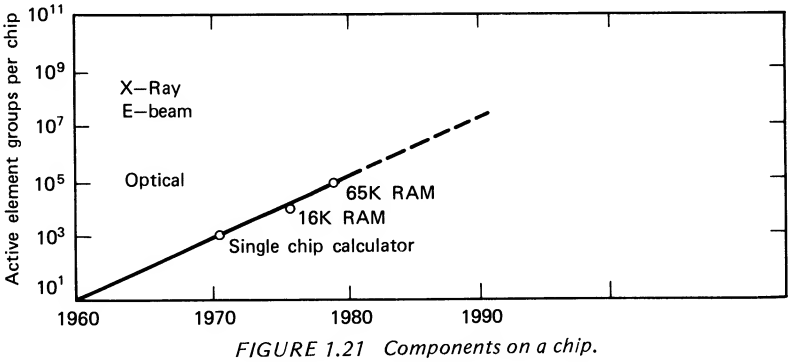


FIGURE 1.21 Components on a chip.

ties. After the laser pulse is turned off, the semiconductor material then almost perfectly recrystallizes.

Lift-off is a technique of defining fine line patterns in evaporated or sputtered layers by the use of a stencil that has a slope at the pattern edge so that a continuous step deposition is not possible. Unwanted material is deposited over a photoresist layer, while the desired pattern is deposited through the stencil openings onto the wafer.

Plasma deposition is the use of a gas plasma to deposit a layer of material originally in the plasma on the semiconductor surface.

Other improvements in semiconductor process technology, such as in ion implantation, make it possible to achieve smaller structures with lower production tolerances and more accurate electrical parameters.

Improvements in Materials

In addition to the trends in silicon semiconductor technology noted above, two other future technological trends should be noted:

- Development work in semiconductor materials other than silicon.
- Research in superconductor technology.

Alternative Semiconductor Materials. For special-purpose applications, there is increasing interest in gallium arsenide as the semiconductor material rather than silicon. The advantages of gallium arsenide (GaAs) are that it is faster and operates at higher temperatures. Some parametric comparisons between the two substrate technologies are shown in Table 1.4.

TABLE 1.4. Silicon Versus Gallium Arsenide

	Silicon	*Gallium Arsenide*
Energy gap (ev)	1.11	1.40
Intrinsic carrier Concentration (cm^{-3})	1.4×10^{10}	9.0×10^{6}
Lattice mobility		
Electrons	1350	8600
Holes	480	250

Superconductor Technology. Two of the most important areas of research in materials technology are superconductor technology and the design of Josephson circuits. Certain materials become "superconductors" (i.e. have zero resistance) when their temperature approaches absolute zero. A persistent current can then be created or destroyed in a region much like a magnetic core can be magnetized or demagnetized, and therefore some of the early research has been directed to memory design with superconductors.

A. basic two-junction Josephson device is shown in Figure 1.22. The principal of operation is that if two superconductors are separated by a thin insulating layer, the layer acts like a superconductor. A magnetic field in the insulator destroys the superconductivity, which results in a detectable voltage drop, indicating the presence of a logical 0 or 1.

It is anticipated that a 16K memory chip using Josephson circuits would have an access time of 15 nsec (nanoseconds), and a read/write cycle time of 30 nsec. Logic circuits have also been developed and present reports indicate a circuit delay of 40 psec (picoseconds) for OR gates and 50–100 psec for AND gates. In fact, it was suggested that a processor containing 300,000 Josephson circuits having a cycle time of 1 nsec could be packaged in an 8 X 8 X 10 cm configuration. Such a processor would include a 256K high-speed memory with a 2-nsec access time, and a 64-megabyte main memory with a 7-nsec access time. The power dissipation for the entire unit would be 10 watts.

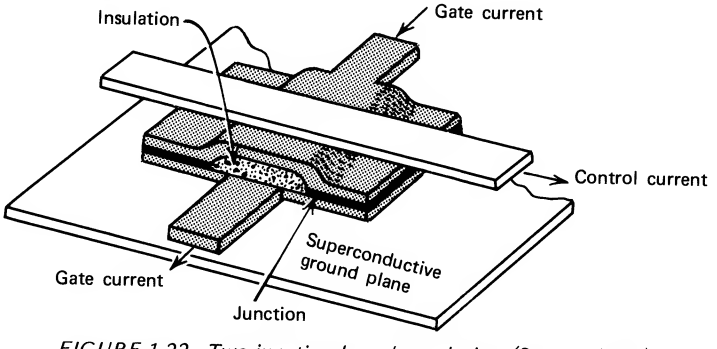


FIGURE 1.22 Two-junction Josephson device. (Source: IBM.)

TABLE 1.5. Evolution of MOS Technology

1972 Enhancement mode *n*-channel	
Channel length	6.0 μ
Junction depth	2.0 μ
Shortest gate delay	12.0 nsec
Speed-power product	18.0 pJ[a]
1976 Depletion mode	
Channel length	6.0 μ
Junction depth	2.0 μ
Shortest gate delay	4.0 nsec
Speed power product	4.0 pJ[a]
1977 HMOS	
Channel length	3.5 μ
Junction depth	0.8 μ
Shortest gate delay	0.5 nsec
Speed power product	0.2 pJ[a]
1980 HMOS-II (projected)	
Channel length	2.0 μ
Junction depth	0.8 μ
Shortest gate delay	0.5 nsec
Speed power product	0.2 pJ[a]

[a]*Pico-Joules.*

Circuit Design

Although significant strides have been made in increasing the maximum economic die size, decreasing the line dimensions, and improving the process technology, the major contribution to the increased number of components on a chip in recent times has been improvements in circuit design and layout.

In recent years circuit design has progressed from use of enhancement mode *n*-channel devices to including depletion load devices, from single level to double level polysilicon, and to development of HMOS technology. Table 1.5 indicates improvements in MOS technology over the past few years with respect to four key parameters.

The Semiconductor Industry

In examining the microcomputer industry, it is important to consider the much broader context of the

TABLE 1.6. Estimated World Sales of Semiconductors by Major International Semiconductor Firms, 1972, 1974, and 1976

		Estimated World Sales (Millions)			*Percentage of Total*		
Company	*Country*	*1972*[a]	*1974*[b]	*1976*[c]	*1972*	*1974*	*1976*
Texas Instruments	United States	$ 405	$ 652	$ 655	11.8	13.7	11.4
Motorola	United States	310	482	462	9.0	10.1	8.0
IBM	United States	275	(c)	(c)	8.0	—	—
Western Electric	United States	195	(c)	(c)	5.7	—	—
Philips	Netherlands	175	277	260	5.1	5.8	4.5
Fairchild	United States	165	324	307	4.8	6.8	5.3
Toshiba	Japan	145	170	233	4.2	3.6	4.0
Hitachi	Japan	135	195	240	3.9	4.1	4.2
ITT	United States	95	160	141	2.8	3.4	2.5
RCA	United States	80	(c)	182	2.3	—	3.2
Siemens	Germany	80	142	(c)	2.3	3.0	—
SGS-ATES	Italy	75	78	(c)	2.2	1.6	—
National Semiconductor	United States	75	204	263	2.1	4.3	4.6
Mitsubishi	Japan	70	80	94	2.0	1.7	1.6
Matsushita	Japan	70	150	254	2.0	3.2	4.4
General Electric	United States	60	114	113	1.7	2.1	2.0
Secosem	France	50	48	(c)	1.5	1.0	—
Signetics	United States	50	121	125	1.4	2.5	2.2
Intel	United States	45	115	147	1.2	2.4	2.6
Fujitsu	Japan	40	(c)	(c)	1.2	—	—
NEC	Japan	—	120	343	—	2.5	6.0
Telefunken	Germany	—	37	(c)	—	0.8	—
General Instruments	United States	—	63	106	—	1.2	1.8
Other companies		855	1841	1834	24.0	34.3	31.8
Total		3450	5,373	5762	100.0	100.0	100.0

SOURCE: U.S. Congress, Senate, Committee on Commerce, Science and Transportation, Industrial Technology, *Hearings, 95th Con., 2d sess., October 30, 1978 (Washington, D.C., U.S. Govt. Print. Off., 1978).*

[a]*U.S. Department of Commerce.*

[b]*Dataquest, Inc.*

[c]*Not available.*

semiconductor industry and to place microcomputers (i.e., microprocessors and associated devices) within the broader industry.

Table 1.6 gives the major international semiconductor firms, and Table 1.7 indicates the market share of the largest firms in the United States in 1975. Integrated circuits only form a portion of all semiconductor devices, but it is a major portion, as shown in Table 1.8, which gives the actual and estimated sales of the various classes of semiconductors over the period 1977–1981 by U.S. based companies. It is particularly noted that of all types of integrated circuits, digital MOS devices account for the most shipments.

The broader categories of discrete devices and integrated circuits can next be analyzed on a geographic basis–In which groups of countries are the products sold? Table 1.9 provides such a geographic summary.

Another important analysis that must be considered is the forecast of product shipments by type of product–digital bipolar, digital MOS, or linear. Before presenting these forecasts of the Semiconductor Industry Association, it is important to list the economic assumptions of the forecast, and the role of dollar devaluation. These assumptions are given in Table 1.10, and the forecasts themselves are presented in Table 1.11.

A statistical analysis of solid state products produced by the semiconductor industry offers only one perspec-

TABLE 1.7. Market Share of U.S. Semiconductor Firms, 1975

Motorola, Inc.	21.3
Texas Instruments, Inc.	15.9
RCA	9.2
Intel	6.4
Teledyne	4.0
Western Electric	3.8
General Electric	3.7

SOURCE: Federal Trade Commission. Staff Report on the Semiconductor Industry, January 1977.

TABLE 1.8. World Semiconductor Shipments of U.S. Based Companies (million $)

Product Summary	*1977*	*1978*	*1979 Est.*	*1980 Est.*	*1981 Est.*
Discrete devices					
Diodes	167	178	169	174	176
Small signal transistors	318	317	304	302	300
Power transistors	313	347	359	375	407
Rectifiers	226	271	271	289	309
Thyristors	146	158	159	173	189
Optoelectronics	155	209	228	260	285
All other discrete	68	78	77	82	87
Total discrete devices	1394	1558	1567	1655	1753
Integrated circuits					
Digital bipolar	766	966	1023	1145	1263
Digital MOS	1136	1484	1751	2119	2626
Linear	561	697	782	927	1098
Total integrated circuits	2463	3147	3556	4191	4987
Total solid state	3857	4705	5123	5846	6740
Percent change					
Discrete		11.8	0.6	5.6	6.0
Integrated circuits		27.8	13.0	18.0	19.0
Total worldwide		22.0	9.0	14.0	15.0

SOURCE: Semiconductor Industry Association. Reprinted by permission.

TABLE 1.9. World Semiconductor Shipments of U.S. Based Companies (million $)

Geographic Summary	*1977*	*1978*	*1979 Est.*	*1980 Est.*	*1981 Est.*
Discrete devices					
United States					
OEM	687	735	730	775	822
Distributor	201	235	236	251	267
Total	888	970	966	1026	1089
Western Europe	340	408	417	434	459
Japan	34	38	38	39	40
Other International	132	142	146	156	165
Total discrete devices	1394	1558	1567	1655	1753
Integrated circuits					
United States					
OEM	1345	1622	1838	2193	2654
Distributor	339	466	524	625	746
Total	1684	2088	2362	2818	3400
Western Europe	545	741	838	959	1102
Japan	130	192	215	250	289
Other international	104	126	141	164	196
Total integrated circuits	2463	3147	3556	4191	4987
Solid State					
United States					
OEM	2032	2357	2568	2968	3476
Distributor	540	701	760	876	1013
Total	2572	3058	3328	3844	4489
Western Europe	885	1149	1255	1393	1561
Japan	164	230	253	289	329
Other International	236	268	287	320	361
Total solid state	3857	4705	5123	5846	6740
Percent change total solid state					
United States					
OEM		16.0	9.0	15.6	17.0
Distributor		30.0	8.4	15.2	15.6
Total		19.0	9.0	15.5	17.0
Western Europe		30.0	9.0	11.0	12.0
Japan		41.0	10.0	14.0	14.0
Other International		13.6	7.1	11.5	12.8
Total worldwide		22.0	9.0	14.0	15.0

SOURCE: Semiconductor Industry Association. Reprinted by permission.

TABLE 1.10. World Semiconductor Forcast, 1978-1981: Assumptions and Definitions

Economic Assumptions

- No recession in 1979
- Real GNP growth slowing in 1979 but rebounding in 1980
- Real GNP range

	1978	*1979*	*1980*
United States	+3.7 to 4.0%	+2.9 to 3.2%	+3.2 to 4.8%
Europe	+2.5 to 3.0%	+3.0 to 3.4%	+3.3 to 4.3%
Japan	+5.0 to 5.5%	+5.5 to 6.0%	+5.0 to 6.0%

Impact of Dollar Devaluation

- Devaluation increased the dollar value of foreign production and markets in 1978. The dollar value of semiconductor shipments may include exchange rate increases.

  Japan +30%; 1977—265 yen/$; 1978—200 yen/$
  Europe +8%

- Devaluation of dollar will help increase sales of U.S. electronic equipment in foreign markets.
- Assumption. The dollar will not change relative to foreign currencies in 1979 and beyond. Forecast is in 1978 exchange rates.
- Devaluation impact on the total solid state growth rate is estimated at 3-4 percentage points in 1978, which implies an actual growth of approximately 18-19%.

SOURCE: Semiconductor Industry Association. Reprinted by permission.

TABLE 1.11. Forecast of Shipments

	1977	*1978*	*1979*	*1980*	*1981*
		Digital Bipolar (million $)			
Geographic Shipments					
United States					
OEM	435	494	519	580	642
Distributor	123	177	187	213	234
Total	558	671	706	793	876
Western Europe	156	213	234	262	286
Japan	41	64	65	70	79
Other International	11	18	18	20	22
Total Worldwide	766	966	1023	1145	1263
% Change		26.1	5.9	11.9	10.3
Product Shipments					
TTL—Schottky logic	134	239	277	348	414
% Change		78.4	16.0	25.6	19.0
TTL—Standard logic	280	321	301	294	285
% Change		11.6	(6.2)	(2.3)	(3.0)
ECL—Logic	51	61	68	78	88
% Change		19.6	11.5	14.7	12.8
Other—Logic	93	104	100	102	108
% Change		11.8	(3.8)	2.0	5.8
Bipolar memory	208	241	277	323	368
% Change		15.8	14.9	16.6	14.2

TABLE 1.11. (*Continued*)

	1977	*1978*	*1979*	*1980*	*1981*
		Digital MOS (million $)			
Geographic Shipments					
United States					
OEM	653	820	967	1188	1500
Distributor	124	179	210	258	326
Total	777	999	1177	1446	1826
Western Europe	247	343	405	470	555
Japan	55	77	92	112	134
Other International	57	65	77	91	111
Total Worldwide	1136	1484	1751	2119	2626
% Change		30.6	18.0	21.0	23.9
Product Shipments					
Logic	633	740	843	984	1185
% Change		11.0	13.9	16.7	20.4
Memory	503	744	908	1135	1441
% Change		48.0	22.0	25.0	27.0
CMOS RECAP Amount already included in logic and memory	235	254	288	345	417
% Change		8.0	13.4	19.8	21.0
		Linear (millions of dollars)			
Geographic Shipments					
United States					
OEM	256	308	352	425	512
Distributor	93	110	127	154	196
Total	349	418	479	579	698
Western Europe	142	185	199	227	261
Japan	34	51	58	68	76
Other International	36	43	46	53	63
Total Worldwide	561	697	782	927	1098
% Change		24.2	12.2	18.5	18.4
Product Shipments					
Amplifiers	133	160	172	196	225
% Change		20.3	7.5	13.9	14.8
Comparators	21	29	30	33	37
% Change		38.0	3.4	10.0	12.1
Voltage Regulators and References	55	72	82	98	117
% Change		31.0	13.9	19.5	19.4
Data Conversion Circuits	64	104	139	181	229
% Change		62.5	33.6	30.2	26.5
Interface	64	78	96	116	141
% Change		21.9	23.0	20.8	21.5
Special Consumer Circuits	170	195	200	232	267
% Change		14.7	2.6	16.0	15.1
Other Linear	54	59	63	71	82
% Change		9.3	6.8	12.7	15.5

SOURCE: Semiconductor Industry Association. Reprinted with permission.

TABLE 1.12. Technological Levels

Facility maintenance
Computational service
Applications programming
Computer languages
Systems programming
System integration
Hardware/software system
Hardware architecture
Logic design
Circuit design
Device design
Materials

tive. Another way of looking at the industry is to consider the various technological levels between raw material and end-user application. Table 1.12 gives the various levels of technology associated with a typical microcomputer.

The computer and semiconductor industry is not highly vertically integrated, although in recent years efforts have been made in the area of systems programming and languages by a number of semiconductor firms. Such trends are shown in Figure 1.23.

The basic trend according to Figure 1.23 is toward end-user applications. It should be pointed out, however, that it is probably not economically efficient for equipment manufacturers to venture to closely into user applications and support. Even the largest mainframe manufacturers, such as IBM with its Series/1, have preferred to let systems houses or OEMs handle user applications and support.

On the other end of the vertical chart of technological levels there is little likelihood of further vertical integration. Silicon is different from crude oil, and although it is academically enlightening to discuss the design and construction of a computer starting with a bucket of sand, there is really no practical reason for a single company to actually carry out all the processes involved.

The Impact of International Technology

The development and application of technology is becoming increasingly international in scope and is having far ranging impacts on international trade, economics, and politics. In this section we consider a number of different aspects of international technology, particularly with reference to the semiconductor industry:

- International trade.
- International economic issues.
- Technological leadership.
- The Japanese challenge.
- Soviet developments.

The subject of international technology, particularly of semiconductors, must first be placed in perspective with respect to the degree of technological product use. One good measure of a high-technology product in widespread use is the general-purpose computer, and Table 1.13 presents the installed value of general-purpose computers in some of the major world markets.

International Trade Aspects

The international trade aspects of the semiconductor industry comprise one of the more significant features that must be considered. Two different aspects of this trade must be distinguished: "offshore" assembly operations and direct import/export.

Offshore assembly operations refers to the shipment of semiconductor chips to assembly facilities in Southeast Asia or other localities. After assembly (a wire

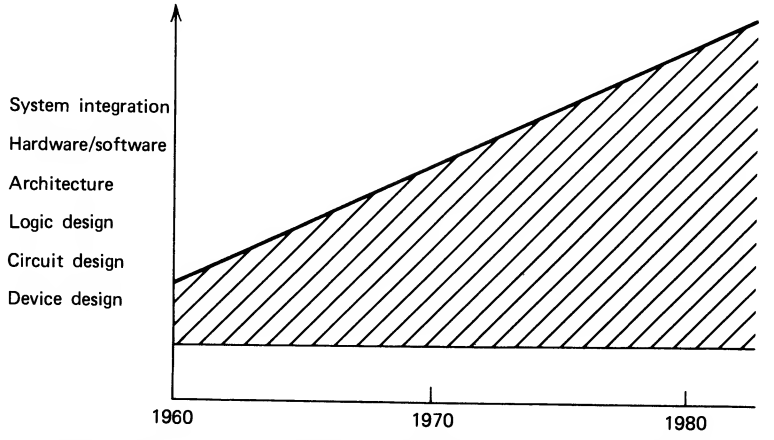


FIGURE 1.23 Vertical integration trends in the semiconductor industry.

TABLE 1.13. Leading General-Purpose Computer Markets

Country	Installed Value of General-Purpose Computer Systems	Percent of Total World Value	World Rank as DP Market
United States	36,100	45.3	1
Japan	8,331	10.0	2
West Germany	6,527	7.9	3
United Kingdom	4,794	5.8	4
France	4,768	5.8	5
Canada	2,301	2.8	7
Italy	2,119	2.6	8
Australasia	2,044	2.5	9
Brazil	532	0.8	17
Middle East	328	0.4	22

SOURCE: International Data Corporation.

bonding and packaging operation), the circuits are imported back into the United States under the provisions of 806.30 and 807.00 of the U.S. tariff law. These tariff provisions state that goods shipped abroad for assembly and then back to the United States are subject to duty only on the value added abroad.

Direct import and export refers to the import or export of completely finished semiconductor devices.

Table 1.14 gives the exports and imports of semiconductors over an 8-year period. This listing includes the export and import trade done under the provisions of 806.30 and 807.00 and thus does not actually represent the international trade in finished semiconductor devices.

Table 1.15 presents the adjusted exports and imports that result from subtracting the 806.30 and 807.00 com-

TABLE 1.14. Total U.S. Exports, Imports, and Apparent Consumption of Semiconductors, 1968–1975 (millions of dollars)

Item	1968	1969	1970	1971	1972	1973	1974	1975
Exports	204.4	345.7	417.0	370.5	469.6	848.5	1247.5	1053.5
Imports	86.4	134.4	189.6	179.1	330.3	618.6	961.3	802.7
Balance of trade[a]	118.0	211.3	227.4	191.4	139.3	229.9	286.2	250.8
Domestic shipments	1414.5	1686.8	1719.7	1501.9	2135.6	3097.0	3249.5	N.A.[c]
Apparent consumption[b]	1206.5	1475.5	1492.3	1310.5	1996.3	2867.1	2963.3	N.A.
Exports as a percent of domestic shipments (%)	14.5	20.5	24.2	24.7	22.0	27.4	38.4	N.A.
Imports as a percent of apparent consumption (%)	6.7	9.1	12.7	13.7	16.5	21.6	32.4	N.A.

SOURCE: Export and import data for 1968–1973 are taken from U.S. Department of Commerce, unpublished Semiconductor Study, Table 4-1. Export and import data for 1974–1975 are taken from Electronic News March 29, 1976, p. 45. Domestic shipments data are taken from Table II-3, Federal Trade Commission, Staff Report on the Semiconductor Industry, January 1977.

[a] *Balance of trade = exports minus imports.*

[b] *Apparent consumption = domestic shipments minus exports plus imports.*

[c] *N.A., not available.*

TABLE 1.15. Adjusted U.S. Exports, Imports, and Apparent Consumption of Semiconductors, 1970–1975 (million $)

Item	1970	1971	1972	1973	1974	1975
Adjusted exports[a]	322.9	263.8	289.9	561.2	927.1	757.7
Adjusted imports[a]	95.5	72.3	150.6	331.4	641.0	506.9
Adjusted trade balance[b]	227.4	191.5	139.3	229.8	286.1	250.8
Domestic shipments	1719.7	1501.9	2135.6	3097.0	3249.5	N.A.[d]
Adjusted apparent consumption[c]	1492.3	1310.4	1996.2	2867.2	2963.4	N.A.
Adjusted exports as a percent of adjusted domestic shipments (%)	18.8	17.6	13.6	18.1	28.5	N.A.
Adjusted imports as a percent of adjusted apparent consumption (%)	6.4	5.5	7.5	11.6	21.6	N.A.

SOURCE: Adjusted exports and imports for 1970–1974 are taken from U.S. Department of Commerce, unpublished Semiconductor Study, Table 4-6; and those for 1975 are from Electronic News, *April 26, 1976; p. 28. Domestic shipments are taken from Table II-3, Federal Trade Commission, Staff Report on the Semiconductor Industry, January 1977.*

[a] *Exports and imports were adjusted by subtracting the U.S. content of tariff items 806.30 and 807.00 from actual exports and imports.*

[b] *Adjusted trade balance = adjusted exports minus adjusted imports.*

[c] *Adjusted apparent consumption = domestic shipments minus adjusted trade balance.*

[d] *N.A., not available.*

TABLE 1.16. U.S. Exports and Imports of Semiconductors from Various Countries, 1975 (million $)

Country	Exports: All Semiconductors	Exports: Finished Semiconductors	Imports
Final product countries			
Belgium and Luxembourg	14.5	12.4	1.3
Canada	28.1	25.5	6.6
France	44.8	32.3	2.3
West Germany	83.9	77.0	13.0
Italy	18.7	16.3	2.5
Japan	67.5	61.4	45.8
Netherlands	8.7	8.2	2.4
United Kingdom	45.2	37.9	7.5
Offshore plant countries			
Brazil	N.A.	N.A.	4.4
Hong Kong	65.3	26.2	71.2
Ireland	9.8	4.8	14.3
Korea	88.8	5.4	104.5
Malaysia	131.9	1.7	161.5
Mexico	121.4	31.6	113.8
Singapore	154.5	9.3	153.9
Taiwan	52.0	13.9	49.4
Other countries	118.4	68.5	48.3
Total	1053.5	432.4	802.7

SOURCE: U.S. Department of Commerce, Domestic and International Business Administration, Bureau of Domestic Commerce, Office of Business Research and Analysis, Science and Electronics Division, unpublished semiconductor study, 1976.

TABLE 1.17. Integrated Circuit Imports to the United States, 1977 (million $)

Malaysia	247
Singapore	230
South Korea	170
Taiwan	82
Phillipines	68
Japan	40
Hong Kong	40
Mexico	40

SOURCE: U.S. Dept. of Commerce.

ponent of the export and import trade. Such a presentation gives a more realistic picture of the balance of trade in semiconductors.

It is also of interest to note the actual countries concerned with the actual import and export trade. Table 1.16 gives both the final product countries (i.e., those concerned with the import or export of finished products) and the offshore plant countries (i.e., those concerned with the import or export of goods in the process of assembly).

Table 1.17 presents more recent data concerning the countries involved in offshore imports into the United States.

In terms of final product countries, Japan is the most serious competitor and has been the subject of significant study by members of the U.S. semiconductor industry. Some of their findings are presented in the section below on the "Japanese challenge."

International Economic Issues

The issue of trade in the semiconductor industry must be placed in perspective in the dynamics of the American economy. This section analyzes some of the economic issues from the broader perspective of the overall U.S. economy, and technology industries operating in that economy.

International trade and technology transfer has become an issue of increasing significance to the American electronics industry. Important structural changes have occured in the U.S. economy and technology industries over the past 10 to 15 years that have had direct consequences on U.S. economic growth, employment, and trade balances. The impact of competitive foreign products, foreign technology, and foreign labor has been considerable in the electronics industry.

A quantitative analysis of those areas of electrotechnology that are particularly affected by foreign technological developments is extremely useful in analyzing the issues posed by increased technology transfer. The present section presents basic data for such an analysis that could be very useful for trade and market studies and allocating research and development resources.

To place the issues of technology transfer in perspective, we must first consider the overall U.S. balance of payments picture.

Table 1.18 shows the U.S. merchandise balance of trade in the years 1960-1977.

TABLE 1.18. United States Merchandise Balance of Trade

Year	Amount (billion $)
1960	$4.9
1965	5.0
1970	2.6
1971	-2.3
1972	-6.4
1973	0.9
1974	-5.3
1975	9.0
1976	-9.4
1977	-31.1

SOURCE: U.S. Council of Economic Advisers, Economic Indicators.

TABLE 1.19. Exports, Imports—Selected Years 1960–1977 (million $)

	1960	1965	1970	1975	1976	1977
All commodities						
Exports	20,408	27,178	42,590	106,102	113,318	117,963
Imports	15,073	21,427	39,952	96,116	120,677	146,817
Balance	5,335	5,751	2,638	9,986	-7,359	-28,854
Electronics apparatus						
Exports	1,090	1,660	2,999	7,582	9,278	10,285
Imports	286	640	2,271	4,911	7,424	8,432
Balance	804	1,020	728	2,671	2,304	1,853

SOURCE: U.S. Department of Commerce, Overseas Business Reports, *"U.S. Foreign Trade."*

TABLE 1.20. U.S. Trade in Technology-Intensive Products (million $)

	1960			1965			1971		
Commodity	*Exports*	*Imports*	*Balance*	*Exports*	*Imports*	*Balance*	*Exports*	*Imports*	*Balance*
Office machinery, including computers	208	68	140	470	136	334	1521	566	955
Electrical machinery	1090	286	804	1660	640	1020	3068	2557	511
Radio, TV, and telecommunications	228	127	101	345	314	31	679	1319	-640
Household electrical appliances	112	9	103	116	27	89	125	158	-33
Electronic tubes and semiconductors	80	13	67	140	63	77	447	259	188
Phonograph and sound equipment	45	24	21	100	107	-7	189	489	-300

SOURCE: U.S. Department of Commerce.

The balance of payments picture presented by Table 1.18 is due to basic structural changes in the U.S. economy. It is not our purpose here to discuss these structural changes or to examine the causative factors. Rather, we merely wish to pinpoint the role of technology-intensive manufactured products in the U.S. balance of payments. Table 1.19 indicates the export and import totals over a period of representative years.

Since the term "electronics apparatus" covers a wide group of products, it would be useful to break out the statistics dealing particularly with some typical electrotechnologies. This is done in Table 1.20.

The trade balance in transistor radios, color televisions, and phonograph equipment was of great concern in the 1960s, as the table indicates. In the 1970s the concern shifted to other products like electronic components where it still could be said that the United States possessed a technological lead. Table 1.21 gives the balance of trade in electronic components, including integrated circuits from 1970 to 1977.

The relative decline of U.S. exports of technology-intensive products coupled with an increase in foreign imports is due to numerous political, economic, and technological factors. Although we are not concerned with the political and economic factors in the present analysis, it should be clearly noted that such factors are much more significant than purely technological factors. Such economic factors include the relative overvaluation of the U.S. dollar in the 1950s and 1960s, different inflation rates in the United States and foreign countries, the development of foreign economic groupings such as the EEC, and the rise of Eurodollar financing.

Technological factors have also been attributed to the unfavorable trends in the balance of payments. Two such major factors include changes in manufacturing productivity and decreasing levels of research and development expenditures.

Productivity is the relationship between the quantity of goods and services produced to the various resources—labor, capital, and so on—that are employed in producing them. Productivity can be measured in many different ways, depending on the purpose for which it is used. One of the most commonly used measurements of productivity relates the total output to the input of labor time. The use of labor productivity in-

TABLE 1.21. U.S. Electronic Components Trade, 1970–1977 (million $)

Year	Total Components	Total Semiconductors	Semiconductor Imports 806/807	Integrated Circuits	Parts of Semiconductors
1970:					
Imports	296	157	139	69	–
Exports	817	417	–	100	167
Balance	+548	+260	–	+31	–
1971:					
Imports	348	179	152	94	–
Exports	747	370	–	91	152
Balance	+399	+191	–	-3	–
1972:					
Imports	520	330	250	180	14
Exports	957	470	–	103	236
Balance	+437	+140	–	-77	+222
1973:					
Imports	877	619	410	357	8
Exports	1670	848	–	218	425
Balance	+783	+229	–	-139	+417
1974:					
Imports	1304	961	682	601	41
Exports	2267	1247	–	313	691
Balance	+963	+286	–	-288	+650
1975:					
Imports	1160	803	617	579	31
Exports	1987	1053	–	262	622
Balance	+827	+250	–	-317	+631
1976:					
Imports	1645	1107	877	808	52
Exports	2532	+1400	–	320	896
Balance	+887	+293	–	-489	+844
1977:					
Imports	2018	1352	1120	1020	63
Exports	2683	1503	–	348	959
Balance	+655	+151	–	-672	+896

SOURCE: U.S. Bureau of the Census.

dices is not meant to suggest that labor is especially responsible for productivity growth, but it is generally used because it is the most easily measured input.

In the United States in the period from 1950 to 1975 productivity grew at an average rate of 2.8% annually in the private business sector. However, the growth rate was different in different industrialized countries, as shown in Table 1.22.

In terms of absolute productivity, it is still believed that the United States continues to have the highest level, though such a statement may not be true for all industries.

The decreasing levels of R & D expenditures in the United States, as a percent of GNP, are presented in Table 1.23.

Perhaps more relevant to our analysis of the semiconductor industry is the R & D expenditures of the

TABLE 1.22. Comparative Growth in Output per Employee-Hour (Average Annual Percentage Change), 1950–1975

United States	2.6
Canada	4.1
France	5.3
Germany	6.0
Japan	9.2
United Kingdom	3.4

SOURCE: Bureau of Labor Statistics U.S. Department of Labor

TABLE 1.23. R & D Expenditures as Percent of GNP

Year	United States	France	Germany	Japan	United Kingdom
1961	2.75	1.38	1.08	N.A.	2.69
1967	2.92	2.16	1.94	1.34	2.75
1973	2.35	1.73	2.36	1.92	N.A.

SOURCE: National Science Foundation.

leading independent United States semiconductor manufacturers. Table 1.24 gives the absolute expenditures of such companies over the years 1972–1978, as well as a percentage of sales.

Technological Leadership

Underlying the general trade and economic issues is the more fundamental issue of technological leadership. There are actually two parts of the leadership issue:

- The public perception of leadership.
- Actual quantified measurements of technological superiority.

The public perception of leadership, rather than some scientifically determined and measured parameter, is probably the most significant factor since it is from such determinations that many purchasing and political decisions are made. Public perceptions can be most accurately measured by polls, and it would be worthwhile to illustrate the change in such perceptions with some highly regarded poll data.

In a poll conducted in five different European countries (Belgium, France, Great Britain, Netherlands, and West Germany) upper middle class households were asked about the following with respect to products manufactured in given European countries, and in the United States:

- The quality of products.
- The value of products.
- The technological advancement of products.

TABLE 1.24. R & D Expenditures of Leading Independent U.S. Semiconductor Manufacturers, 1972–1978 (Million $; Fiscal Years)

Company	*1972*	*1973*	*1974*	*1975*	*1976*	*1977*	*1978*[a]
Texas Instrument:[b]							
Dollars	154.00	185.00	198.00	183.00	187.00	195.00	–
Percent of sales	16.32	14.37	12.59	13.38	11.27	9.53	–
FCI[b]							
Dollars	29.29	38.25	40.29	38.00	47.00	49.70	54.00[c]
Percent of sales	13.08	10.89	10.47	13.03	10.60	10.80	–
NSC:							
Dollars	9.80	11.70	18.70	20.72	24.85	30.71	41.83
Percent of sales	16.39	11.81	8.75	8.80	7.64	7.93	8.46
Motorola:							
Dollars	76.00	95.00	100.25	98.48	96.41	109.73	–
Percent of sales	6.53	6.61	7.22	7.51	66.28	5.94	–
Intel:							
Dollars	3.44	4.56	10.50	14.54	20.71	27.92	41.40
Percent of sales	14.70	6.90	7.81	10.63	9.16	9.88	10.83
AMD:							
Dollars	0.25	0.75	1.49	1.72	2.13	5.53	6.96
Percent of sales	5.39	6.71	5.62	6.64	6.20	8.90	7.54
AMI:							
Dollars	2.37	3.41	6.03	5.79	6.45	6.68	6.82
Percent of sales	8.31	5.87	8.01	8.76	9.60	9.39	8.43
Mostek:							
Dollars	1.10	2.58	4.27	4.14	3.91	4.79	–
Percent of sales	6.21	6.16	7.14	8.79	6.78	5.60	–
Intersil:							
Dollars	1.39	1.70	2.51	1.99	6.81	6.61	7.86
Percent of sales	10.81	5.40	4.10	3.46	9.37	8.04	7.21

SOURCE: Dataquest, Inc., Company 10-K and annual reports, January 1979. Reprinted from U.S. Congress, Senate, Committee on Commerce, Science, and Transportation, Industrial Technology, *Hearings, 95th Cong., 2d sess., October 30, 1978 (Washington, D.C., U.S. Govt. Print. Off., 1978).*

[a] *1978 data is not available until annual reports are received by us.*

[b] *Some R & D is Government funded–for Texas Instruments, about 50 percent, 10 percent for the period covered in this table. For Fairchild, approximately $6,000,000 (1977) $4,000,000 (1976) and $3,000,000 (1975) was funded by outside firms or U.S. Government.*

[c] *Estimated.*

To avoid national bias, the respondents were not asked about products from their own country. The results of the poll, listing the countries in order of the number of households listing that country as leader, is presented in Table 1.25.

TABLE 1.25. Product Rating from Poll Data (%)

Quality	
West Germany	82
Netherlands	25
United States	24
Value	
West Germany	43
Italy	22
United States	21
Technological advancement	
West Germany	68
United States	52

SOURCE: Market and Opinion Research International, London, 1978.

It is worthwhile to attempt to quantify the apparent eroding leadership of the United States in the development of new technology relative to other countries. One approach is by examining the records of the U.S. Patent Office concerning new U.S. patents issued and determining from which countries such patent applications arose.

A patent represents a new and useful process, machine, manufacture or composition of matter, or any new and useful improvement thereof. By studying the proportion of U.S. patents issued to foreign residents we would be able to gauge foreign activity in particular technologies.

Of course, any methodology is subject to qualifications. Patents represent only one form of scientific and technological activity. Pure research, represented by scientific publications, "know-how," trade secrets, and

confidential information represent other forms of industrial property that should be part of any measurement of technological development. Furthermore, there is no predetermined relationship between the application and issuance of a U.S. patent to a foreign resident and the original application for a domestic patent by the foreign resident, representing the same or similar technologies. Such relationships may change over time as a result of domestic economic conditions, firm profitability, marketing strategy, or other variables.

There are, however, a number of positive motivations for choosing patents as a parameter representing technological leadership. A patent is an industrial property right and therefore is more often associated with commercial technology than scientific or military advances. The U.S. market is one of the most important for new technological products. The filing of a foreign origin patent application in the United States is costly, and therefore only important technological advances with potential U.S. use are generally filed.

With such qualifications in mind, we can present the data. Figure 1.24 shows the percentage of all U.S. patents issued to foreign residents.

Although this Figure only shows the percent of U.S. patents issued to foreign residents up to the year 1971, later years show a consistent increase, reaching 35% in 1975. It should be pointed out that in categorization for statistical purposes, the Patent and Trademark Office considers a U.S. patent to be of foreign origin if the first named inventor is a non U.S. resident. Such trends are clear evidence of the increased development of foreign technology, or at least the increased interest of foreign residents in commercially exploiting such technology in the United States.

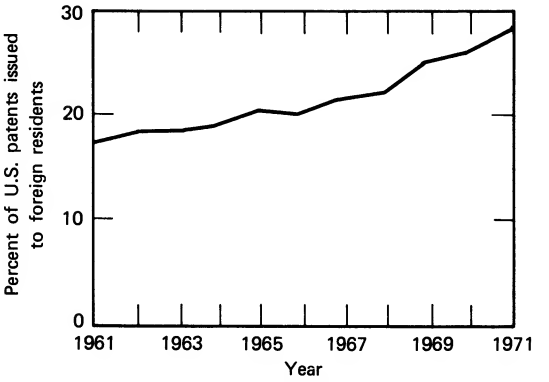


FIGURE 1.24 U.S. Patents issued to foreign residents. (Source: U.S. Patent and Trademark Office.)

Figure 1.25 shows the trend of U.S. patents issued to residents of various foreign countries from 1963 to 1971, while Table 1.26 indicates the leading countries whose residents obtained U.S. patents in 1974.

TABLE 1.26. U.S. Patents Issued to Foreign Residents

Country	*Number*	*Percent*
Germany, Fed. Rep.	6243	7.7
Japan	6116	7.6
United Kingdom	3273	4.0
France	2626	3.2
Switzerland	1484	1.8

SOURCE: U.S. Patent and Trademark Office.

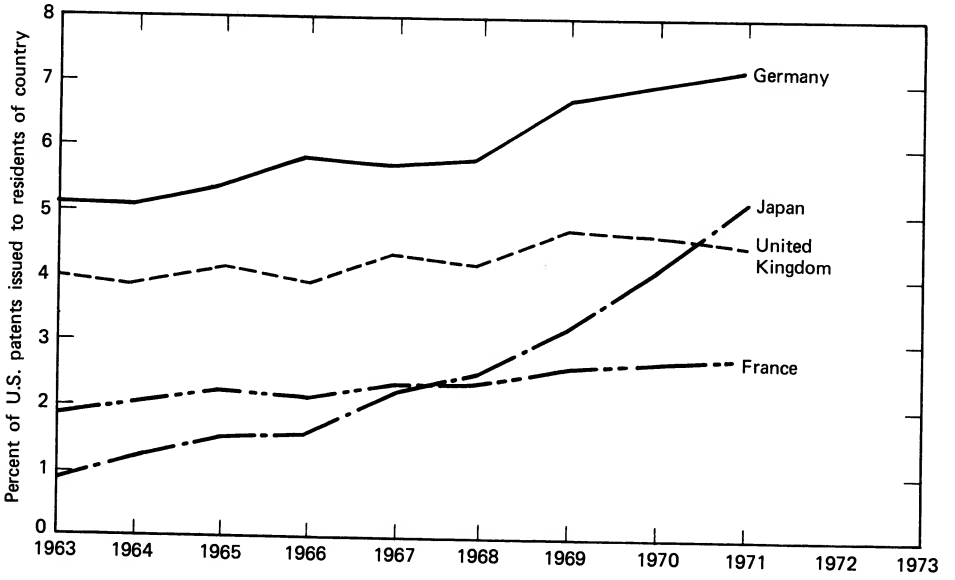


FIGURE 1.25 U.S. Patents issued to residents of specific countries. (Source: U.S. Patent and Trademark Office.)

A more detailed analysis of technological activity is presented by considering the broad categories of technology represented by the classes of the U.S. Patent Office Classification System. Table 1.27 presents the classes dealing with the electrotechnologies and identifies their class numbers. Certain classes have been excluded from the table and the profiles that follow because of insignificant patent activity.

Figure 1.26 provides a gauge on overall technological activity within each of the classes. The number appearing within or adjacent to the bar associated with each class number is the number of U.S. patents issued in

TABLE 1.27. U.S. Patent Office Classification System

Classification Number	*Subject Matter*
313	Electric lamp and discharge devices (e.g., grids, anodes, cathodes, lamp envelopes, spark plugs)
315	Electric lamp and discharge devices, systems (e.g., cathode ray tube circuits)
317	Electricity (e.g., safety and protection systems; polarity reversing, igniting, switchboard devices; circuits and relays, barrier layer devices and capacitors)
318	Electricity, motive power systems (e.g., electric motors, servo systems)
320	Electricity, battery and condenser charging and discharging
321	Electricity conversion systems (e.g., rectification, phase and frequency conversion)
322	Electricity, single generator systems
323	Electricity, voltage magnitude and phase control systems
324	Electricity, measuring and testing
325	Modulated carrier wave communication systems
328	Miscellaneous electron space discharge device systems (e.g., pulse generators)
329	Demodulators and detectors
330	Amplifiers
331	Oscillators
332	Modulators
333	Wave transmission lines and networks
334	Tuners
335	Electricity, magnetically operated switches, magnets and electromagnets
336	Inductor devices
337	Electricity, electrothermally or thermally actuated switches
338	Electrical resistors
339	Electrical connectors
340	Communications, electrical (e.g., data processing, traffic control, telemetry systems)
343	Communications, radio wave
346	Recorders (includes apparatus for recording the movement of machines and various phenomena, e.g., workmen's time clock, seismic cameras)
350	Optics, systems and elements
351	Optics, eye examining, vision testing and correcting
352	Optics, motion pictures
353	Optics, image projectors
355	Photocopying
356	Optics, measuring and testing (e.g., range finders, photometers, spectroscopes, etc.)

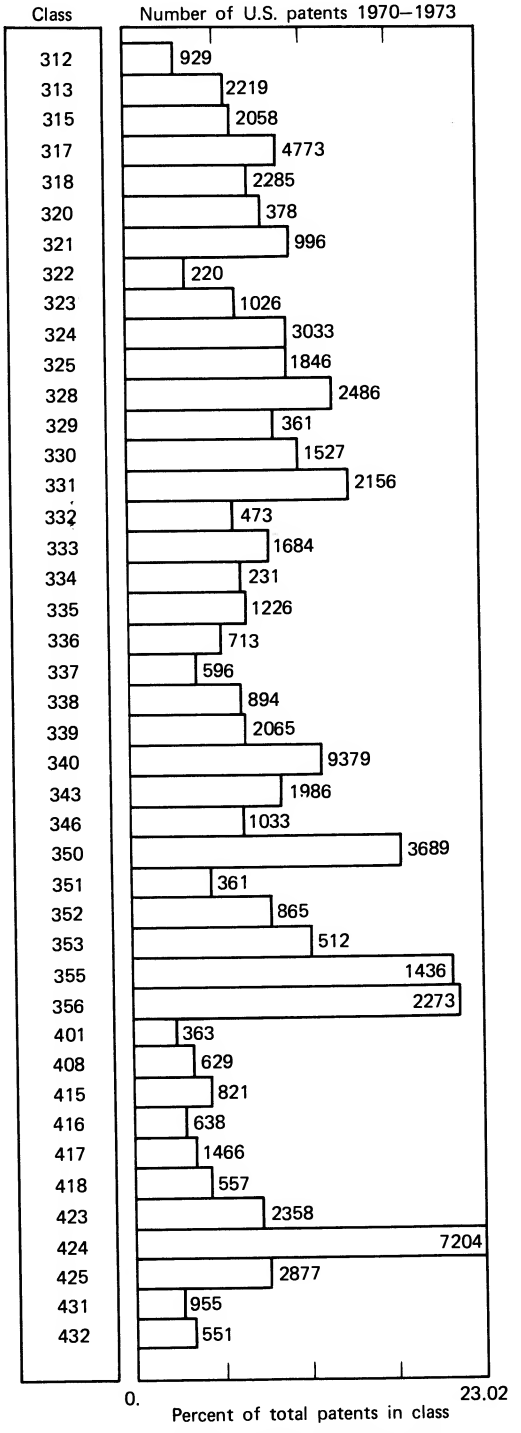


FIGURE 1.26 U.S. Patents issued 1970–1973 by technology class. (Source: U.S. Patent and Trademark Office.)

that class in the 3-year period ending June 30, 1973. For example, class 317, Electricity, had 4773 patents issued in the above mentioned 3-year period.

The length of the bar in Figure 1.26 represents the ratio of the number of patents issued in the 3-year sample period to each hundred total patents in the class. The bar length thus serves as a means for indicating the percentage growth of the class that the recent patents represent, adjusting for the disparities in class sizes. Thus at a glance one is able to determine the fastest growing areas of technology.

Figure 1.27 represents technological activity of U.S. residents, while Figure 1.28 represents that of foreign residents. The numbers in the chart and the length of the bar have the same meaning as they did for Figure 1.26. A comparison of these two figures immediately pinpoints the areas of technology that are receiving increased emphasis, both in the United States and abroad. An examination of the ratios of the numbers given in Figures 1.27 and 1.28 indicate the relative importance of foreign technology in any particular area.

With the methodology developed above it is possible to ask more specific questions dealing with particular technologies defined by patent subclasses. For example, a search may be made of patent subclasses to determine those areas of electrotechnology having the highest percentage of patents issued to foreigners over the 10-year period 1963-1972. Such an analysis would pinpoint those technologies in which foreign companies have a particularly strong position. The results are indicated in Table 1.28.

A closer examination of trends within each technological area is also possible to further elucidate those areas with the greatest increase of foreign activity. The areas of electrotechnology having the fastest growth of foreign technology are given in Table 1.29.

The figures in this table deserve some comment and qualification. First, the above areas represent those specific technologies in which foreign activity has been most evidenced and are not necessarily the most important technology. Second, increases in foreign patent activity alone should not be taken as conclusive evidence of foreign lead in that particular technology.

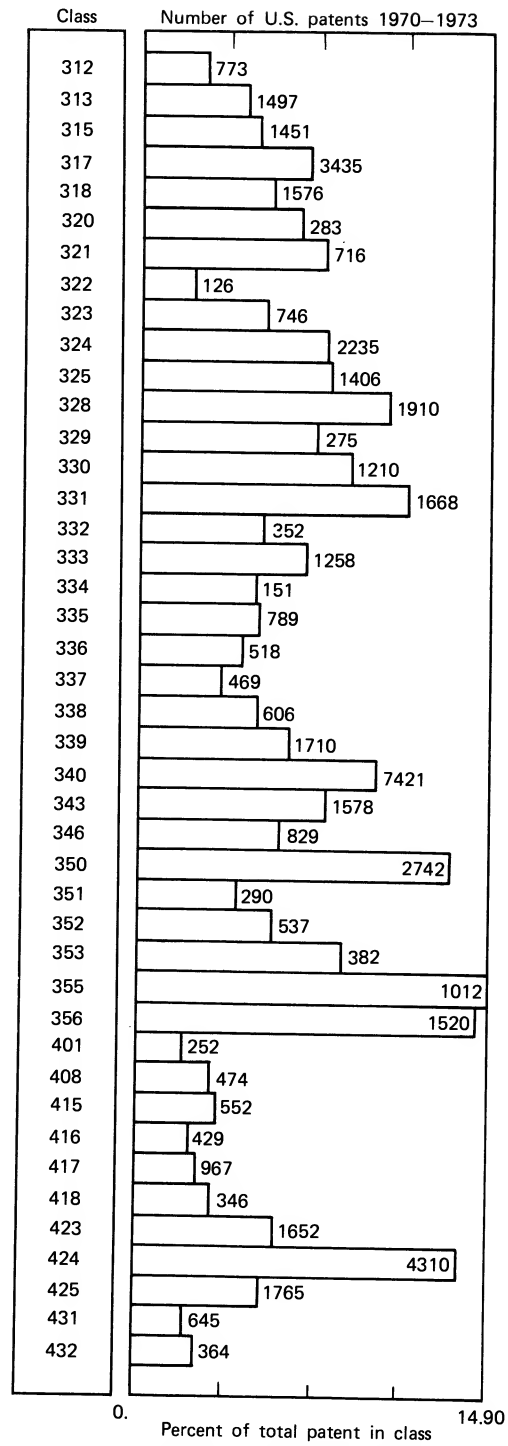


FIGURE 1.27 U.S. Patents issued 1970–1973 by technology class to U.S. residents. (Source: U.S. Patent and Trademark Office.)

TABLE 1.28. Electrotechnology Areas Having Highest Foreign Share, 1963–1972

	Percent Patents Foreign
Magnetic field responsive resistors	72
Superconductors	60
Semiconductor internal structure	52
Magnetic sound recording and reproducing apparatus	52
Ignition timing control for internal combustion engines	49

SOURCE: U.S. Patent and Trademark Office.

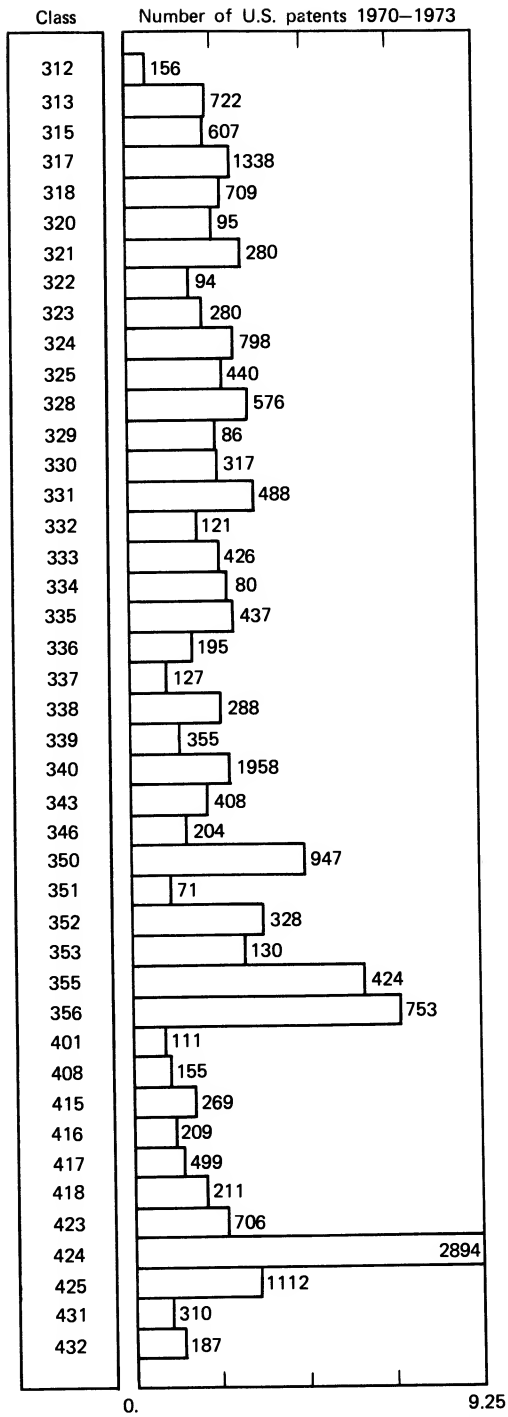


FIGURE 1.28 U.S. Patents issued 1970–1973 by technology class to foreign residents. (Source: U.S. Patent and Trademark Office.)

One situation in which the statistics may be subject to incorrect interpretation is the case where scientific or market interest in a particular technology "peaks" over a certain period of time. Although U.S. firms may divert research and development efforts away from a disproven or unpopular technology, foreign firms may not as yet have developed sufficient background in the technology. As a result, the number of U.S. patents being issued may decline, while foreign patents increase.

TABLE 1.29. Electrotechnology Areas Having Highest Projected Rate of Increase of Foreign Share

	Projected Foreign Share (%)
Magnetic field responsive resistors	90
Electronic musical instruments	70
Telephone recording systems	70
Analysis of complex electrical signals	44
Electrical high-frequency filters	42

SOURCE: U.S. Patent and Trademark Office.

This built-in technology transfer "lag" also arises in part because foreign origin patent applications are filed in the U.S. about a year after they are initially filed abroad. Thus foreign patents issue from the U.S. patent office at a correspondingly later time after preparation of the initial patent application.

Another factor concerns the nature of the patents themselves. Mere quantitative information on the number of patents issued is not necessarily a reliable guide to basic technological development. In some cases the first few patents in a technological area are basic, fundamental patents that are extremely valuable industrial property. Such patents may be followed by a much larger number of "improvement" patents that are mere variations of the basic patents and that have been secured to obtain a position in an important technology. Thus the number of patents in themselves is not an indication of their value or of technological leadership.

In any event, such broad classifications are only useful for a theoretical economic analysis. An industrial technology transfer assessment is only concerned with a few specific technology subclasses. In such subclasses, it would be desirable to have the most detailed information. As an example of such a detailed analysis, we present the analysis of the subclass concerning semiconductor internal structure, a highly important area of technological activity both in the United States and abroad.

The technological development in semiconductor internal structure, as measured by patents issued in the United States, is presented in Table 1.30 a detailed breakdown by year and by country. The pertinent activity indices in terms of percent are presented in Table 1.31.

Although the discussion above uses data from the late 1960s and early 1970s, recent data show similar trends. In addition to merely considering foreign versus U.S.

TABLE 1.30. Distribution of Origin of U.S. Patents Concerning Semiconductor Internal Structure[a]

Country	*1963*	*1964*	*1965*	*1966*	*1967*	*1968*	*1969*	*1970*	*1971*	*1972*	*Total 1963–1972*
Japan	1	1	2	6	17	7	20	8	10	22	94
Germany	1	6	4	13	17	15	23	5		3	87
United Kingdom		3	4	3	11	11	7	3	1	1	44
Netherlands	3		3	1	7	3	6	1	2	2	28
France		1		1	2	6	3	2	1	1	17
Canada		1			3	6	1	1	1	1	14
Sweden			1		2	2	3			1	9
Switzerland	1			1	1		3		1	1	8
Denmark				1	1	1	1	1			5
Italy		1					2				3
U.S.S.R.						1	1				2
Spain			1								1
Norway						1					1
Total foreign	6	13	15	26	61	53	70	21	16	32	313
United States	37	35	53	94	132	160	163	42	28	28	772
Total	43	48	68	120	193	213	233	63	44	60	1085

SOURCE: U.S. Patent and Trademark Office.

[a]Subject matter related to the internal structure (e.g., crystal orientation, zone structure, faults, doping, concentrations, ion implantation) of a region (e.g., base, emitter) of a semiconductor device.

TABLE 1.31. Activity Indices: Semiconductor Internal Structure

Average percent foreign	52
Projected average percent foreign	65
Minimum percent U.S. owned of foreign	11
Average percent growth	2.8
Average percent assigned to U.S. government	0.3

SOURCE: U.S. Patent and Trademark Office.

activity in specific patent classes, it is also of interest to identify the specific companies active in various technological areas.

It is not surprising that in 1977 IBM Corporation led with the greatest number of patents in the class on electronic computers and data systems (class 364), or that Westinghouse and General Electric led in the class on electric systems and devices (class 361). But it may come as a surprise that the leader in number of patents issued in the class on active solid state devices was Hitachi Ltd.

The semiconductor industry is not alone among American industries in being challenged by increased foreign imports, and the semiconductor industry is also not alone is showing evidence of relatively decreasing technological activity in key patent classes. Take the automobile industry, for example, which prides itself on engineering. Such pride does not explain why in the class of internal-combustion engine patents in 1977 the leaders were Toyota, then Nissan Motor, and Robert Bosch GmbH.

The above review is not presented as a definitive or conclusive analysis of technology leadership, but only to demonstrate a methodology that could be useful in a professional technology assessment or long range planning study for a specific technology or product. Such efforts should necessarily include a closer examination and evaluation of each individual patent and the specific trade flows associated with each technology or product. Such studies could be directed at pinpointing particular needs or technological capabilities throughout the world markets. The availability of source material from the U.S. Patent Office has greatly simplified such a task.

The Japanese Challenge

The penetration of Japanese industry into many areas of the American economy has been widely noted. More recently, Japanese companies have begun to enter the U.S. semiconductor industry, and considerable concern has arisen among industry leaders with respect to the fate of the industry while the Japanese "challenge" is still that, and not Japanese "dominance."

Before we consider the actual charges and countercharges made in the industry, it would be worthwhile to place the subject of Japanese trade in perspective.

Figure 1.29 indicates the imbalance in U.S./Japan trade. Such an imbalance has resulted in the sharp increase in the value of the yen with respect to the U.S. dollar, as international monetary markets attempt to adjust to such disparities. The relative price of the yen and the U.S. dollar are shown in Figure 1.30.

More directly relevant are the trade statistics for the

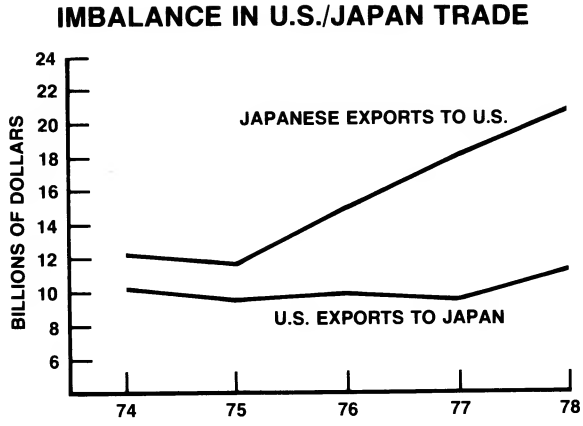


Figure 1.29 Imbalance in U.S./Japan trade. (Source: National Semiconductor Corporation.)

Japanese semiconductor and integrated circuit industry. Figure 1.31 shows the Japanese semiconductor industry's production, export, and import of discrete devices. Figure 1.32 shows the Japanese industry's production, exports, and imports of integrated circuits. For both this figure and Figure 1.31 the Semiconductor Industry Association (SIA) in the United States has provided data on the amount of goods exported by the United States to Japan.

Further and more recent detail on Japanese semiconductor production is presented in Table 1.32 and on Japanese exports in Table 1.33.

The Japanese semiconductor production and export data must be placed in the perspective of the relative market share between Japanese and U.S. companies,

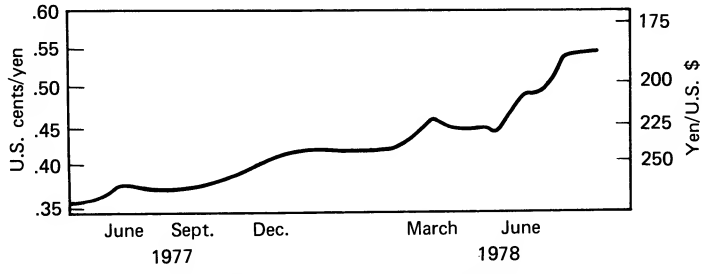


FIGURE 1.30 Value of Japanese yen.

JAPANESE S/C INDUSTRY
DISCRETE DEVICES
PRODUCTION – EXPORT – IMPORT

UNITS – MILLIONS DOLLARS

YEAR	PRODUCTION	IMPORT	EXPORT	DOMESTIC DEMAND	IMPORT PENETRATION
1971	$ 414M				
1972	583				
1973	896	$50M	$ 78M	$868M	5.7%
1974	739	50	105	684	7.3%
1975	529	36	79	486	7.4%
1976	857	80	157	780	10 %
1977	1080	87 (SIA/36)	206	961	9 %

SOURCE: MITI
1977 – DATAQUEST – LARGEST INCREASE DUE TO INCREASED VALUE OF YEN

FIGURE 1.31 Japanese semiconductor industry. (Source: National Semiconductor Corporation.)

JAPANESE IC INDUSTRY
IC PRODUCTION – IMPORT, DOMESTIC DEMAND & IMPORT PENETRATION

UNITS – MILLIONS DOLLARS AND %

YEAR	PRODUCTION	IMPORT	EXPORT	DOMESTIC DEMAND	IMPORT PENETRATION
1971	$120M	67M	0M	187M	35.9%
1972	197	55	1	251	21.8%
1973	364	128	10	482	26.5%
1974	366	170	22	514	33.1%
1975	349	133	44	438	30.5%
1976	587	209	75	721	29.0%
1977	751	146 $\left(\frac{SIA}{131}\right)$	107	790	18.5%

SOURCE: KIKAI TOKEI
CUSTOMS DEPARTMENT JAPANESE GOVERNMENT
DEMPA SHAMBUN

FIGURE 1.32 Japanese IC industry. (Source: National Semiconductor Corporation.)

TABLE 1.32. Japanese Semiconductor Production

Year	*Value (Million Yen)*	*Percent Change from Previous Quarter*	*Value (Million) ($ = 300Y)*	*Percent Change from Previous Quarter*	*Approximate Current Exchange Rate[2] (Yen per Dollar)*	*Value (Million) Using Current Exchange Rate*	*Percent Change from Previous Quarter*
1976:							
1st quarter	95,690	–	$319.0	(Same as yen)	300	$317.9	–
2d quarter	108,651	13.8	362.1		298	364.5	14.7
3d quarter	124,434	14.4	414.8		292	425.1	16.9
4th quarter	125,476	.9	418.3		290	432.7	1.5
1977:							
1st quarter	123,624	(1.5)	412.1		284	435.3	.6
2d quarter	118,330	(4.3)	394.4		272	435	(.1)
3d quarter	122,711	3.7	409		265	463.1	6.5
4th quarter	118,177	(3.7)	393.9		251	470.8	1.7
1978:							
1st quarter	124,334	5.2	414.4		230	540.6	14.8
2d quarter	125,910	1.3	419.7		205	614.2	13.6
3d quarter	141,809	12.6	472.7		190	764.4	21.5
Net change, 1st quarter 1976 to 3d quarter 1978	–	48.2	–		–	–	134.8

SOURCE: MITI Dataquest, Inc., January 1978. Reprinted from U.S. Congress, Senate, Committee on Commerce, Science, and Transportation, Industrial Technology, *Hearings, 95th Cong., 2d Sess., October 30, 1978 (Washington, D.C., U.S. Govt. Print. Off., 1978).*

TABLE 1.33. Japanese Semiconductor Production and Exports—1976 to 2nd Quarter of 1978 (All Values in Billion Yen)

Year	Production	Total Exports	Percent of Production	Exports to United States	Percent of Productio
1976:					
1st quarter	95.7	[a]	[a]	[a]	[a]
2d quarter	108.7	[a]	[a]	[a]	[a]
3d quarter	124.4	15.1	12.1	2.7	2.2
4th quarter	125.5	17.2	13.7	3.6	2.8
1977:					
1st quarter	123.6	17.5	14.2	3.2	2.6
2d quarter	118.3	18.7	15.9	4.6	3.9
3d quarter	122.7	18.0	14.7	4.4	3.6
4th quarter	118.2	18.5	15.7	5.2	4.4
1978:					
1st quarter	124.3	18.4	14.8	5.3	4.3
2d quarter	125.9	21.9	17.4	5.8	4.6
3d quarter	141.8	22.2	15.7	5.9	4.2

SOURCE: MITI, Dataquest, Inc., January 1978. Reprinted from U.S. Congress, Senate, Committee on Commerce, Science, and Transportation, Industrial Technology, *Hearings, 95th Cong., 2d Sess., October 30, 1978 (Washington, D.C., U.S. Govt. Print. Off., 1978).*

[a]*Not available.*

as well as the consumption of semiconductor components in such countries. Table 1.34 gives the total integrated circuit market share estimates for the United States, Japan, and Europe, and Table 1.35 presents the consumption of semiconductor components in such countries.

Another important assessment is United States and Japanese trade in semiconductor products with each other. Table 1.36 assembles the available data from different sources concerning United States, Japanese, and European trade with each other.

Figure 1.33 shows the Japanese exports of semiconductor products to the United States and the U.S. exports to Japan in 1976. The figures represent not only the estimated values of semiconductor devices sold separately as components, but also those incorporated into finished equipment before sale. They indicate that Japan apparently ships more semiconductors to the United States than the United States ships to Japan.

TABLE 1.35. Estimated Consumption of Semiconductor Components (Millions $)

Country	1973	1974	1975	1976	1977	1978
United States	$1964	$2266	$1810	$2353	$2712	$3262
Japan	1156	1276	996	1531	1623	1708
Europe	1336	1591	1333	1592	1773	2078

SOURCE: U.S. Congress, Senate, Committee on Commerce, Science, and Transportation, Industrial Technology, *Hearings, 95th Cong., 2d Sess., October 30, 1978 (Washington, D.C., U.S. Govt. Print. Off., 1978).*

Although the relative importance of Japanese semiconductors in the U.S. market is not that significant at present, there is increasing concern that governmental policies may unfairly give Japanese companies certain advantages that over the long run may cause the Japanese domination of some U.S. semiconductor markets much like their domination over other U.S. industries, such as steel, motorcycles, high-fidelity electronics, television receivers, and various passive electronic components.

It must be emphasized that the U.S. companies are not concerned with competition per se. There is no doubt that the U.S. semiconductor industry is highly competitive and based on the concept of free and open trade.

It is the governmental policies that are the cause of concern, namely:

- The 12% import tariff in Japan versus 6% in the U.S.
- The highly restrictive distribution and marketing

TABLE 1.34. Total Integrated Circuit Market Share Estimates (Millions $)

Sector	1970	1971	1972	1973	1974	1975	1976	1977
U.S. companies	$625	$663	$883	$1450	$18.75	$1559	$2083	$2577
Percent of total market	75.6	72.6	70.9	73.5	75.2	73.7	70.6	72.9
Japanese companies	$143	$153	$241	$343	$372	$343	$599	$661
Percent of total market	17.3	16.8	19.9	17.3	14.9	16.2	20.3	18.7
European companies	$53	$87	$107	$161	$210	$204	$260	$288
Percent of total market	6.4	9.5	8.6	8.1	8.4	9.6	8.8	8.1
Rest of world	$6	$10	$14	$23	$38	$10	$8	$10
Percent of total market	0.7	1.1	1.1	1.1	1.5	0.5	0.3	0.3
Total market	$827	$913	$1245	$1986	$2495	$2116	$2950	$3536

SOURCE: U.S. Congress, Senate, Committee on Commerce, Science, and Transportation, Industrial Technology, *Hearings, 9th Cong., 2d Sess., October 30, 1978 (Washington, D.C., U.S. Govt. Print. Off., 1978).*

TABLE 1.36. Estimated Trade in Semiconductor Components (Millions $)

Countries	*Souce*	*1973*	*1974*	*1975*	*1976*	*1977*	*1978*
United States to Europe	SIA	$591	$795	$630	$756	$885	$1149
United States to Japan	SIA	122	130	102	168	164	230
Japan to United States	MITI/DQ	20	38	56	54	85	174[a]
Japan to Europe	MITI/DQ	NA[b]	NA	NA	NA	64	70[a]
Europe to United States	—	NA	NA	NA	NA	NA	NA
Europe to Japan	MITI	NA	NA	NA	NA	44	38[a]
United States to Japan	MITI	92	115	90	134	142	193

SOURCE: SIA, MITI, Dataquest, Inc. Reprinted from U.S. Congress, Senate, Committee on Commerce, Science, and Transportation, Industrial Technology, *Hearings, 95th Cong., 2d Sess., October 20, 1978 (Washington, D.C., U.S. Govt. Print. Off., 1978).*

[a] *Preliminary.*

[b] *Not available.*

Note: Semiconductor Industry Association (SIA) data is sales by U.S. firms to applicable destination, not necessarily exports. Much of U.S. sales to Europe, and some U.S. sales to Japan are produced by U.S. companies in region of buyer. Japanese exports calculated from MITI data with 47 percent adder. MITI data translated from yen at 300 yen equals $1, 1970–76; 265 yen equals $1, 1977; and 195 yen equals $1, 1978.

COMPARISON –
JAPANESE SEMICONDUCTOR EXPORTS TO U.S.
WITH U.S. EXPORTS TO JAPAN – 1976

JAPAN TO U.S.

C.I.F. VALUE OF SEMI:	**$ 49,670,000**
SEMI VALUE/CONTENTS IN END PRODUCT:	**$185,710,000**
CY 1976 TOTAL:	**$235,380,000**

U.S. TO JAPAN

C.I.F. VALUE OF SEMI:	**$167,000,000**
SEMI VALUE/CONTENTS IN END PRODUCT:	**$ 50,000,000**
CY 1976 TOTAL:	**$217,000,000**

SOURCE: U.S. CUSTOMS
JAPANESE CUSTOMS

FIGURE 1.33 Comparison Japanese/U.S. semiconductor exports. (Source: National Semiconductor Corporation.)

facilities in Japan, making it difficult for foreign companies to enter and compete in the Japanese market.

- The close relationship between business and the government in Japan, such as the VLSI program and the establishment of a VLSI laboratory in Kawasaki in cooperation with leading Japanese companies.
- Financing and funding advantages of Japanese companies that U.S. companies lack.

The Japanese counter argument to these concerns, as pointed out in a recent report of the U.S./Japan Trade Agreement, is to politely note that the problem is not amenable to "simple solution."

Perhaps the Japanese response is also part of the problem. The deeply ingrained social, institutional, moral, and legal attitudes in Japan are totally different from those in America. It is as difficult to objectively assess the true cost of a product manufactured in Japan as it is to determine the cost of golf carts manufactured in Poland solely for export. The economic systems simply do not permit easy assessments. With no objective economic cost data, it is impossible to make an argument for what is fair and reasonable.

Soviet Developments

There is increasing interest in developments in microcircuit technology in the Soviet Union. It would there-

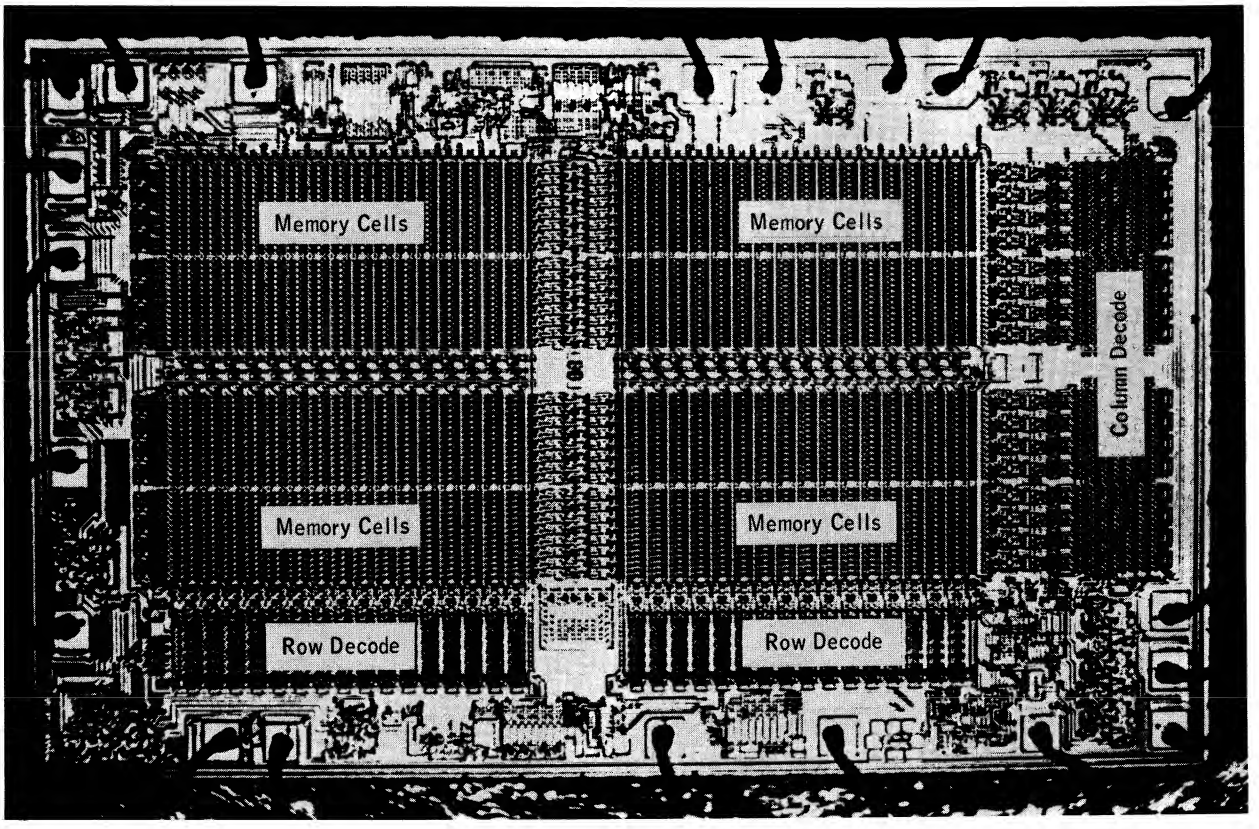


FIGURE 1.34 Soviet 4K dynamic random access memory. (Copyright 1979 Aviation Week and Space Technology. *Reprinted by permission.)*

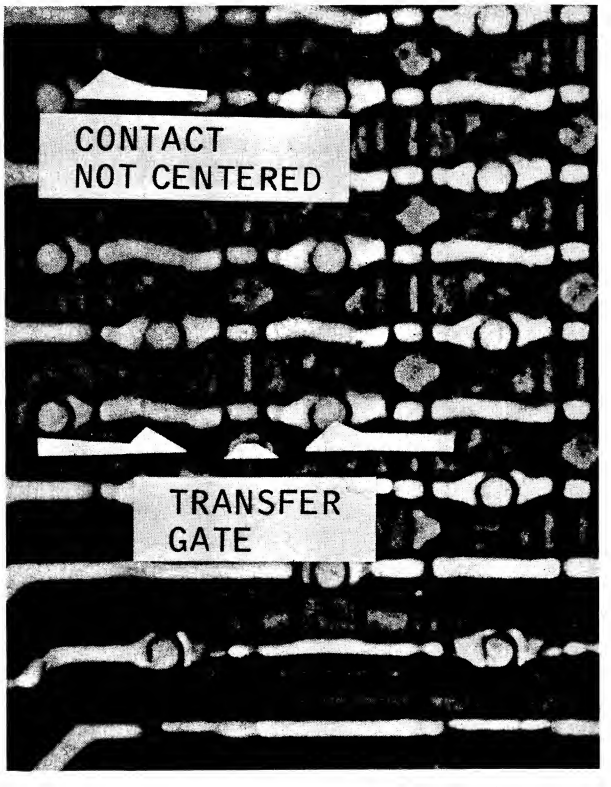


FIGURE 1.35 Soviet 16K dynamic RAM close-up (Copyright 1979 Aviation Week and Space Technology. *Reprinted by permission.)*

fore be worthwhile to conclude the present chapter with an examination of some recent Soviet microcircuits.

The Soviet Union presently produces a 4-bit microprocessor implemented in CMOS technology, a CMOS peripheral controller chip, and a variety of mask-programmable ROMs, also in CMOS. Also produced are 4K and 16K RAMs using *n*-channel silicon gate MOS technology.

The 4K memory, shown in Figure 1.34, is packaged in ceramic with a cover of brazed metal.

The 16K memory, an enlarged portion of which is shown in Figure 1.35, is packaged in a similar manner. The specification indicates a read cycle time of 510 nsec minimum, and a supply voltage of 12.0 V ± 5%. Some of the memory cell features, including a contact that was not precisely centered over the *n*+ well below, are shown in Figure 1.35.

References

Chang, Y. S. *The Transfer of Technology: Economics of Offshore Assembly, the Case of the Semiconductor Industry*. New York, UNITAR, 1971.

National Academy of Engineering. *U.S. Technology and International Trade*. Washington D.C., National Academy of Sciences, 1976.

National Research Council. *Technology, Trade, and the U.S. Economy*. Washington D.C., National Academy of Sciences, 1978.

National Science Board. *Science Indicators 1974*. Washington D.C., National Science Foundation, 1974.

U.S. Congress. Senate. Committee on Commerce, Science, and Transportation. *Industrial Technology*. Hearings, 95th Cong., 2d sess. October 30, 1978. Washington D.C., U.S. Govt. Print. Off., 1978.

U.S. Department of Commerce. Patent and Trademark Office. Office of Technology Assessment and Forecast. *Reports*. 1973–1979.

U.S. Department of Labor. Bureau of Labor Statistics. *Productivity and the Economy*. Bulletin 1926. Washington D.C., U.S. Govt. Print. Off., 1977.

U.S. Federal Trade Commission. Bureau of Economics. *Staff Report on the Semiconductor Industry*. January 1977.

2

the sixteen-bit microprocessors

The new 16-bit microprocessors announced in 1978 are one of the most important developments in the brief history of the microcomputer industry. The capability of handling 16-bit data and address words has extended the capability of the microprocessor to that of the minicomputer, opening up many new applications. In this chapter, we overview the architecture of some of the representative 16-bit microprocessors:

- Intel 8086.
- Zilog Z8000.
- Texas Instruments 9900.
- Texas Instruments 9940.
- Fairchild 9440.
- Motorola 68000.
- Ferranti F 100L.
- Data General mN 602.
- National 16000 family.
- Advanced Micro Devices 29116.

We also examine some 8-bit microprocessors with 16-bit internal architecture, such as the Intel 8088, Motorola 6809, and National 16008.

Intel 8086

The Intel 8086, introduced in 1978, is a 16-bit microprocessor fabricated using the *n*-channel Intel HMOS process. The functional block diagram of the 8086 is shown in Figure 2.1, and the pin diagram of the 40-pin package is shown in Figure 2.2. A photomicrograph of the Intel 8086 is presented in Figure 2.3.

The architecture of the 8086 can be described in terms of the block diagram shown in Figure 2.1. The operations of the processor are divided into two basic units: a bus interface unit (BIU) and an execution unit (EU). These units generally are separate and distinct processors that operate asynchronously.

The BIU is concerned with instruction fetching and

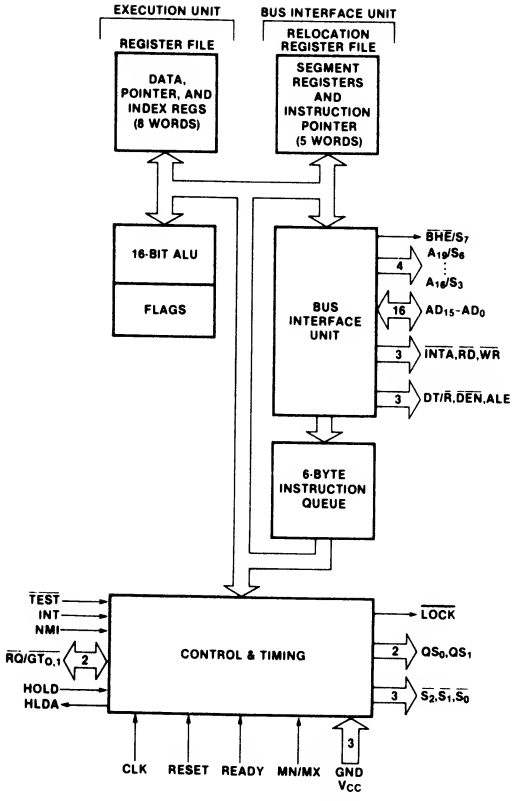


FIGURE 2.1 Intel 8086 functional block diagram. (Source: Intel Corporation.)

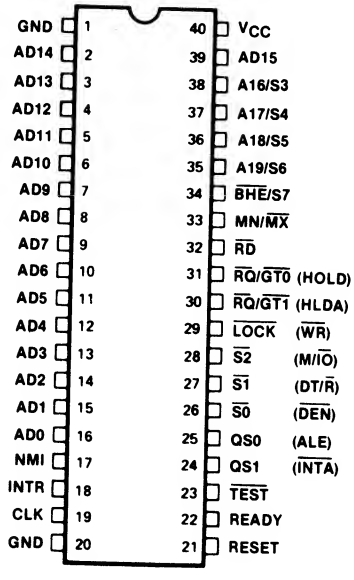


FIGURE 2.2 Intel 8086 pin diagram. (Source: Intel Corporation.)

queuing, operand fetch and store, and address relocation. The 6-byte instruction queue acts as a first-in-first-out buffer (FIFO), from which the EU is able to service instruction bytes as required. The instruction queue permits instruction prefetching, thereby increasing the performance of the processor. The BIU also contains segment registers and an instruction pointer.

The EU contains the register file, flags, and the 16-bit CPU. The EU functions to process prefetched instructions from the BIU, and results are passed to the BIU for storage.

The 8086 has 20 address pins, thus permitting a 20-bit memory address and direct addressability of 1 megabyte. Words (each word consists of 2 bytes or 16-bits) can be located on either even or odd address boundaries. In referencing word data, the BIU requires either one or two memory cycles depending on whether the starting byte of the word is on an even or an odd address. The BIU automatically performs the proper number of memory accesses, and operates transparently to the user. By "transparently" we mean that the BIU operates independently of the user and without being immediately noticeable to the user.

The 8086 has a time multiplexed data and address bus to make use of the standard 40-pin package. I/O ports are addressed in the same manner as memory locations. The 8086 can address up to 64K I/O byte registers, or 32K I/O word registers.

Further details on the system configuration, timing, and instruction set of the 8086 presented in Chapter 3.

Zilog Z8000

The Zilog Z8000 is a 16-bit *n*-channel MOS microprocessor that was introduced in 1979. The functional block diagram of the Z8000 is shown in Figure 2.4, and the pin diagram is shown in Figure 2.5.

The Z8000 is available in two configurations: segmented and nonsegmented, as indicated in the pin diagram in Figure 2.5. The segmented version operates with an external memory management device that allows variable segment sizes, memory protection, and relocation features. The segmented version can directly access up to 8 megabytes of memory. The segmented version is implemented in a 48-pin package, while the nonsegmented version is implemented in a 40-pin package.

Memory segmentation in the Z8000 provides for the designation of up to 128 segments that reference areas of memory variable in size from 256 bytes to 64 kilobytes, in increments of 256 bytes.

The Z8000 has sixteen 16-bit registers in addition to special system registers. All sixteen 16-bit registers

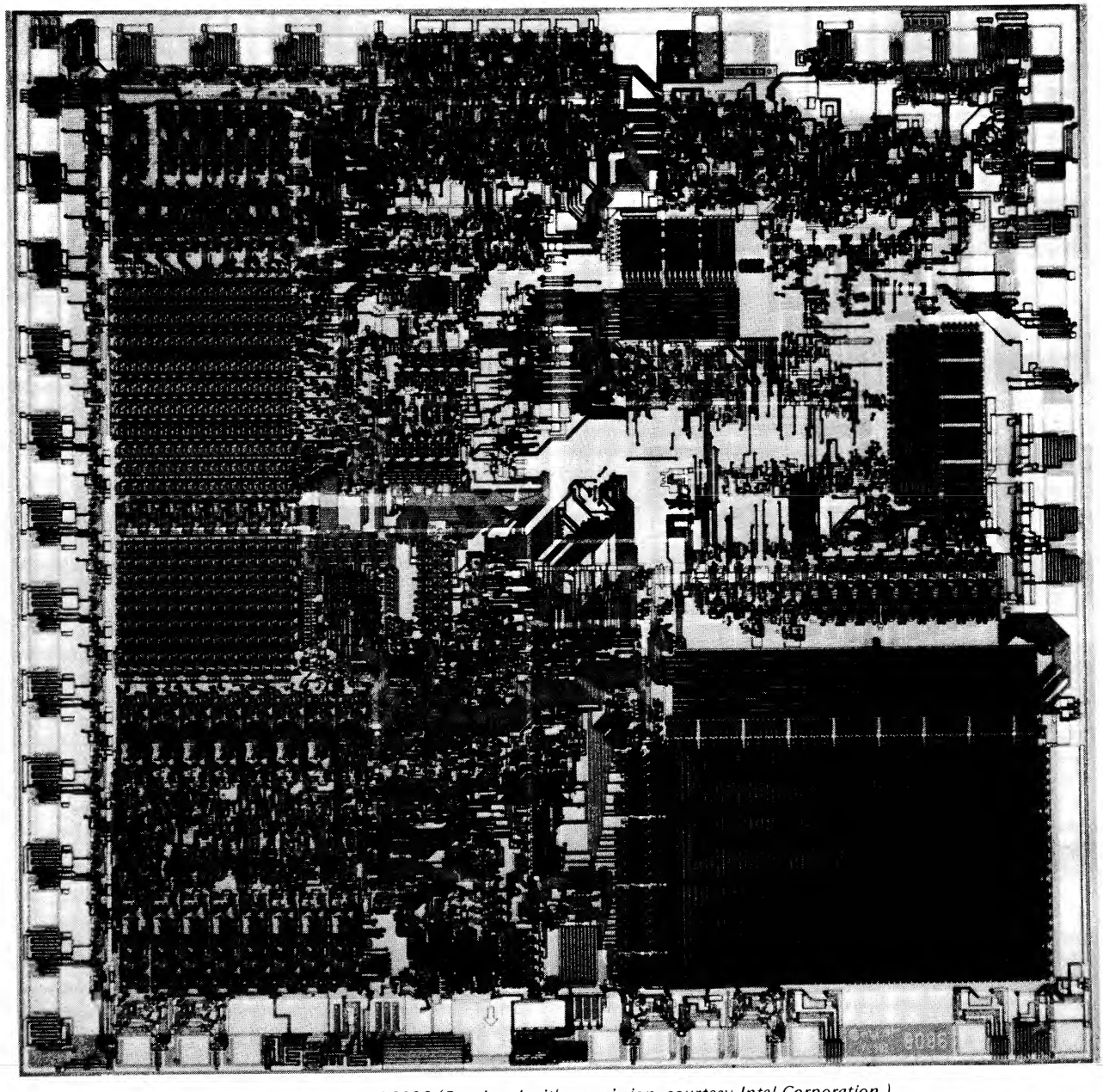


FIGURE 2.3 Intel 8086 (Reprinted with permission, courtesy Intel Corporation.)

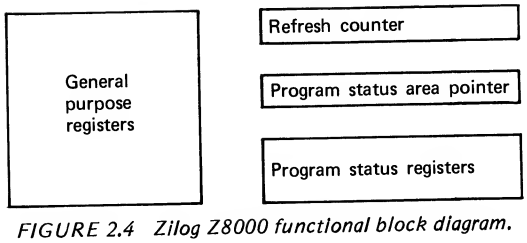


FIGURE 2.4 Zilog Z8000 functional block diagram.

may be used as accumulators, and all but one can serve as index registers.

Seven main data types are supported: bits, BCD digits, bytes, words (16-bits), long words (32-bits), byte strings, and word strings. Other data elements are memory addresses, I/O addresses, segment table entries, and program status words (PSW).

The Z8000 features over 110 distinct instruction types. Each instruction can generally use any of five main addressing modes: register (R), indirect register (IR), direct address (DA), indexed (X), and immediate (IM). For certain instructions there are several other addressing modes: base address (BA), base indexed (BX), realtive address (RA), autoincrement, and autodecrement.

Figure 2.6 shows Z8000 execution times for different

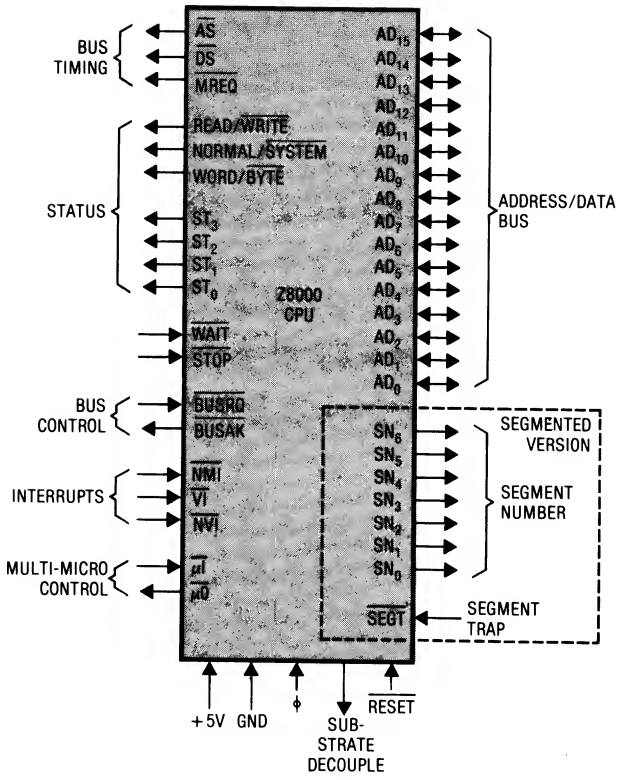


FIGURE 2.5 Zilog Z8000 pin diagram. (Source: Zilog, Inc. Reproduced by permission. © 1979 Zilog, Inc. This material shall not be reproduced without the written consent of Zilog, Inc.)

addressing modes for the common instruction LDB R, which is to load the B register with the contents of register R. Comparison is made with a standard minicomputer, the PDP 11/45. It is seen that the Z8000 is slightly faster than the PDP 11/45. The results are given in Table 2.1.

TABLE 2.1. Execution Times for LDB R, src (μsec)

Source Addressing Mode	*Z8000 at 4 MHz*	*PDP 11/45 With 8K*
Register	0.75	0.90
Indirect register	1.75	1.88
Direct addressing	2.25	2.78
Indexed addressing	2.50	2.78
Immediate	1.00	1.88

SOURCE: Zilog, Inc. Reproduced by permission. © 1979 Zilog, Inc. This material shall not be reproduced without the written consent of Zilog, Inc.

Further benchmarks of the PDP 11/45 versus the Z8000 for different instructions and for different data element types are presented in Figure 2.7. The addressing mode used in Figure 2.7 is direct addressing, abbreviated DA after the actual instruction. The different instructions considered are (*a*) loading, (*b*) adding, and (*c*) multiplying. The results presented in Figure 2.7 are given in Table 2.2.

Further details on the Zilog Z8000 are presented in Chapter 5.

Texas Instruments TMS9900

The TMS9900 is a 16-bit *n*-channel MOS microprocessor that was first introduced in 1975. Although the TMS9900 was one of the earlier 16-bit microprocessors to be introduced, it remains important because of the development of a 9900 family of microprocessor products that are continuously being introduced by the manufacturer, Texas Instruments. This family includes the following:

- SBP9900, a version of the TMS9900 using I^2L

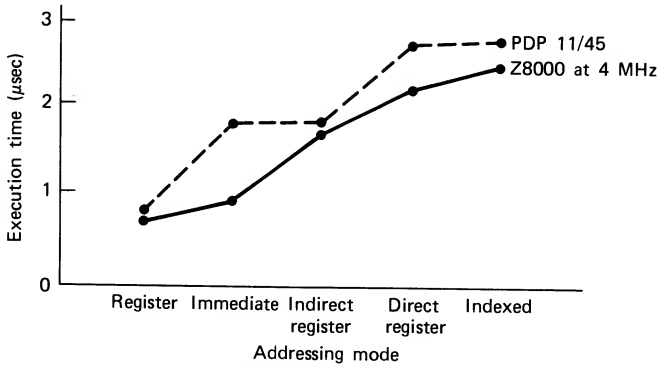


FIGURE 2.6 Zilog Z-8000 execution times by addressing mode. (Source: Zilog, Inc. Reproduced by permission. © 1979 Zilog, Inc. This material shall not be reproduced without the written consent of Zilog, Inc.)

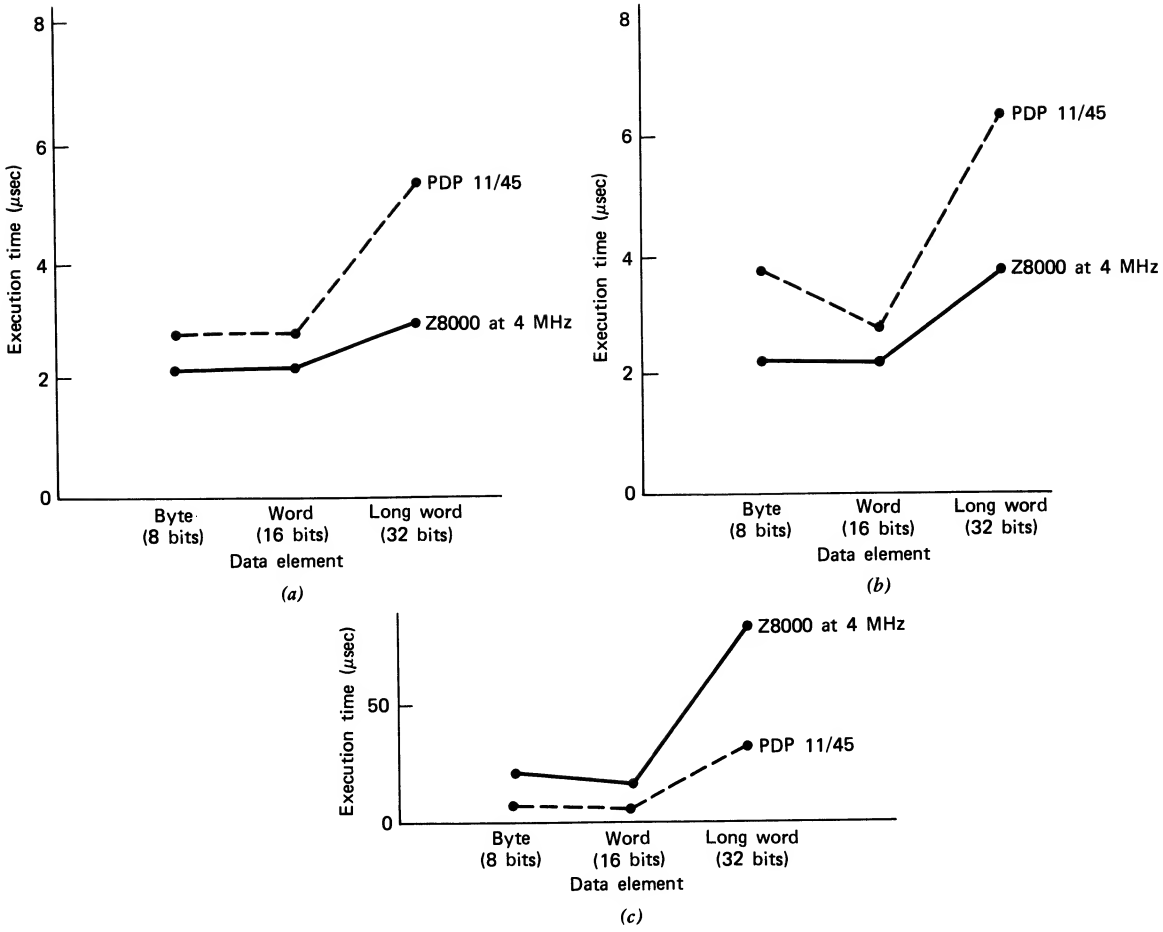


FIGURE 2.7 Zilog Z8000 execution times by data element for (a) *LD R, DA;* (b) *ADD R, DA;* (c) *MULT R, Da. (Source: Zilog, Inc. Reproduced by permission. © 1979 Zilog, Inc. This material shall not be reproduced without the written consent of Zilog, Inc.)*

TABLE 2.2. Execution Times for LD, ADD and MULT.

		Z8000 at 4 MHZ				*PDP 11/45 With 8K*		
Operation	*Data Type*	*Inst.*	*Bytes*	*Cycles*	*μsec*	*Inst.*	*Bytes*	*μsec*
LD R, DA	Byte	1	4	9	2.25	1	4	2.78
	Word	1	4	9	2.25	1	4	2.78
	Long word	1	4	12	3.00	2	8	5.56
ADD R, DA	Byte	1	4	9	2.25	2	6	3.68
	Word	1	4	9	2.25	1	4	2.78
	Long word	1	4	15	3.75	3	10	6.46
MULT R, DA	Byte	3	8	87	21.75	2	6	6.61
	Word	1	4	70	17.50	1	4	5.56
	Long word	1	4	≈350	≈88	17	42	33.94[a]

SOURCE: Zilog, Inc. Reproduced by permission. © 1979 Zilog, Inc. This material shall not be reproduced without the consent of Zilog, Inc.

[a] *If double floating point is used, it is one instruction, 4 bytes and ≈7.23 μsec.*

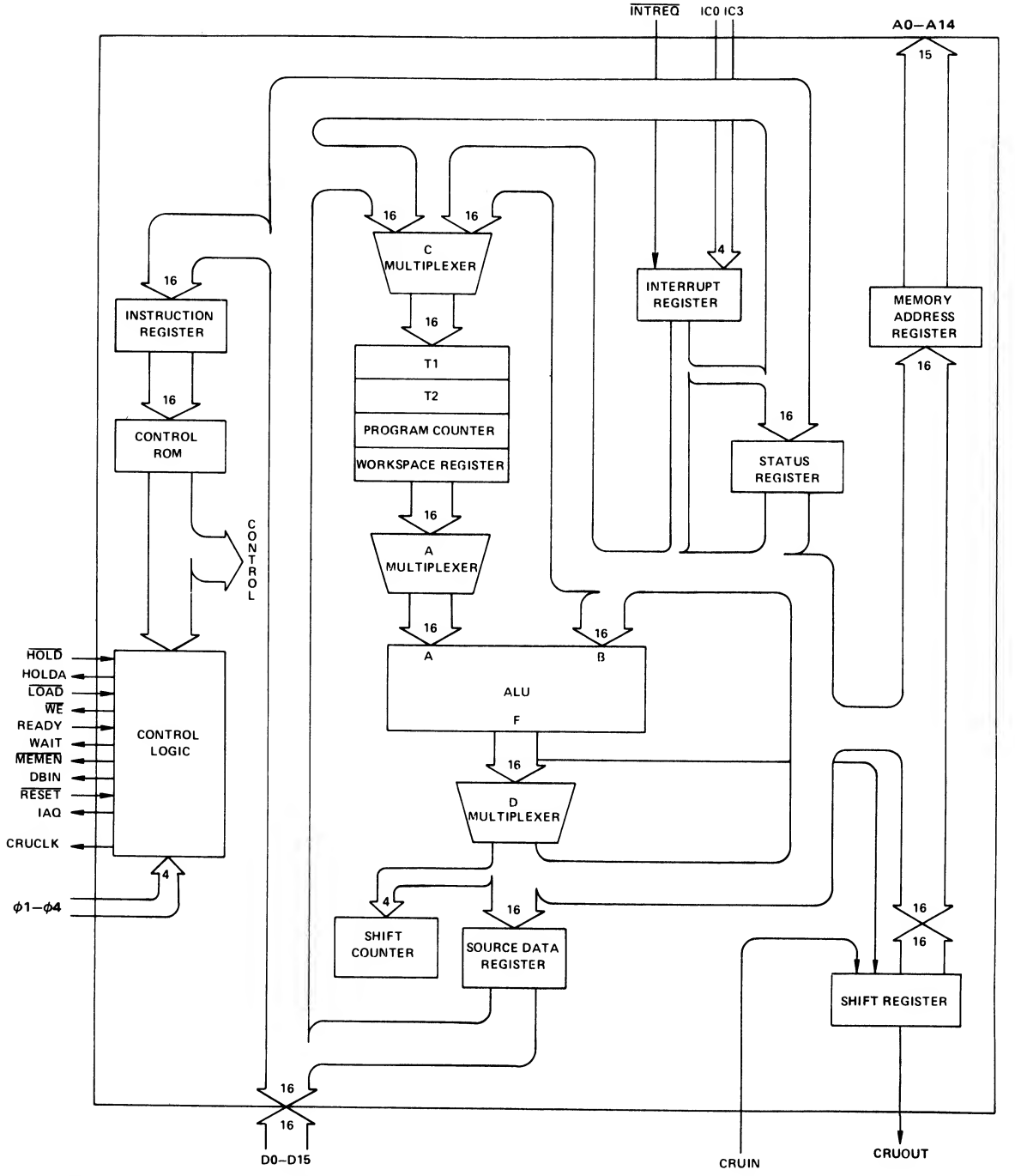


FIGURE 2.8 Texas Instruments TMS 9900 functional block diagram. (Source: Texas Instruments Incorporated.)

technology designed to handle military temperature ranges.

- TMS9980, an 8-bit data bus version of the TMS9900 for smaller systems.
- TMS9940, a version of the TMS9900 incorporating 128 bytes of RAM on-chip, and 2K bytes of mask-programmable or reprogrammable ROM on-chip for single-chip, low-cost controller application.

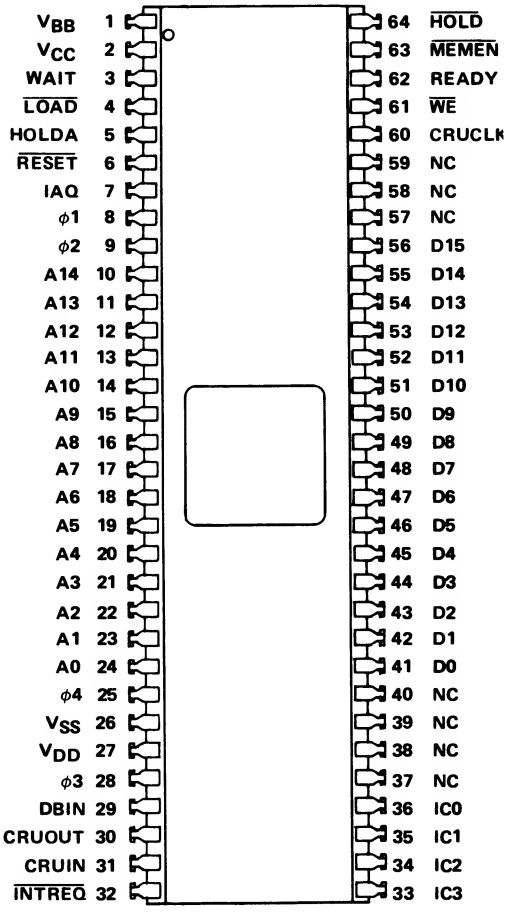
Also related to the 9900 family of microprocessors is the Texas Instruments 990 family of minicomputers. The 990/4 microcomputer is a single board computer incorporating the TMS9900 as its central processing unit. The 990/10 minicomputer is a TTL implementation of the 990/4 for high performance in many applications.

Returning to the TMS9900 itself, the functional block diagram of the chip is shown in Figure 2.8 and the pin

TMS 9900 PIN ASSIGNMENTS AND FUNCTIONS

SIGNATURE	PIN	I/O	DESCRIPTION
			ADDRESS BUS
A0 (MSB)	24	OUT	A0 through A14 comprise the address bus.
A1	23	OUT	This 3-state bus provides the memory-
A2	22	OUT	address vector to the external-memory
A3	21	OUT	system when $\overline{\text{MEMEN}}$ is active and I/O-bit
A4	20	OUT	addresses and external-instruction addresses
A5	19	OUT	to the I/O system when $\overline{\text{MEMEN}}$ is inactive.
A6	18	OUT	The address bus assumes the high-impedance
A7	17	OUT	state when HOLDA is active.
A8	16	OUT	
A9	15	OUT	
A10	14	OUT	
A11	13	OUT	
A12	12	OUT	
A13	11	OUT	
A14 (LSB)	10	OUT	
			DATA BUS
D0 (MSB)	41	I/O	D0 through D15 comprise the bidirectional
D1	42	I/O	3-state data bus. This bus transfers memory
D2	43	I/O	data to (when writing) and from (when
D3	44	I/O	reading) the external-memory system when
D4	45	I/O	$\overline{\text{MEMEN}}$ is active. The data bus assumes the
D5	46	I/O	high-impedance state when HOLDA is
D6	47	I/O	active.
D7	48	I/O	
D8	49	I/O	
D9	50	I/O	
D10	51	I/O	
D11	52	I/O	
D12	53	I/O	
D13	54	I/O	
D14	55	I/O	
D15 (LSB)	56	I/O	
			POWER SUPPLIES
V_{BB}	1		Supply voltage (−5 V NOM)
V_{CC}	2		Supply voltage (5 V NOM)
V_{DD}	27		Supply voltage (12 V NOM)
V_{SS}	26		Ground reference
			CLOCKS
$\phi 1$	8	IN	Phase-1 clock
$\phi 2$	9	IN	Phase-2 clock
$\phi 3$	28	IN	Phase-3 clock
$\phi 4$	25	IN	Phase-4 clock

TMS 9900 PIN ASSIGNMENTS



NC — No connection

FIGURE 2.9 Texas Instruments TMS 9900 pin diagram. (Source: Texas Instruments Incorporated.)

diagram and functional description are given in Figure 2.9.

The most significant feature of the TMS9900 architecture shown in Figure 2.8 is that there are no internal general-purpose registers. The TMS9900 uses a memory-to-memory architecture that locates such registers in memory. Only three registers in the TMS9900 are accessible to the user: the program counter (PC), the status register (ST), and the workspace pointer (WP).

Blocks of memory in a TMS9900 system are designated as a "workspace" and replace the internal registers found in other microprocessors. All data operations with the TMS9900 take place by fetching a word from a portion of the workspace, performing the operation

SIGNATURE	PIN	I/O	DESCRIPTION
			BUS CONTROL
DBIN	29	OUT	Data bus in. When active (high), DBIN indicates that the TMS 9900 has disabled its output buffers to allow the memory to place memory-read data on the data bus during MEMEN. DBIN remains low in all other cases except when HOLDA is active.
MEMEN	63	OUT	Memory enable. When active (low), MEMEN indicates that the address bus contains a memory address.
WE	61	OUT	Write enable. When active (low), WE indicates that memory-write data is available from the TMS 9900 to be written into memory.
CRUCLK	60	OUT	CRU clock. When active (high), CRUCLK indicates that external interface logic should sample the output data on CRUOUT or should decode external instructions on A0 through A2.
CRUIN	31	IN	CRU data in. CRUIN, normally driven by 3-state or open-collector devices, receives input data from external interface logic. When the processor executes a STCR or TB instruction, it samples CRUIN for the level of the CRU input bit specified by the address bus (A3 through A14).
CRUOUT	30	OUT	CRU data out. Serial I/O data appears on the CRUOUT line when an LDCR, SBZ, or SBO instruction is executed. The data on CRUOUT should be sampled by external I/O interface logic when CRUCLK goes active (high).
			INTERRUPT CONTROL
INTREQ	32	IN	Interrupt request. When active (low), INTREQ indicates that an external interrupt is requested. If INTREQ is active, the processor loads the data on the interrupt-code-input lines IC0 through IC3 into the internal interrupt-code-storage register. The code is compared to the interrupt mask bits of the status register. If equal or higher priority than the enabled interrupt level (interrupt code equal or less than status register bits 12 through 15) the TMS 9900 interrupt sequence is initiated. If the comparison fails, the processor ignores the request. INTREQ should remain active and the processor will continue to sample IC0 through IC3 until the program enables a sufficiently low priority to accept the request interrupt.
IC0 (MSB)	36	IN	Interrupt codes. IC0 is the MSB of the interrupt code, which is sampled when INTREQ is active. When IC0 through IC3 are LLLH, the highest external-priority interrupt is being requested and when HHHH, the lowest-priority interrupt is being requested.
IC1	35	IN	
IC2	34	IN	
IC3 (LSB)	33	IN	
			MEMORY CONTROL
HOLD	64	IN	Hold. When active (low), HOLD indicates to the processor that an external controller (e.g., DMA device) desires to utilize the address and data buses to transfer data to or from memory. The TMS 9900 enters the hold state following a hold signal when it has completed its present memory cycle. The processor then places the address and data buses in the high-impedance state (along with WE, MEMEN, and DBIN) and responds with a hold-acknowledge signal (HOLDA). When HOLD is removed, the processor returns to normal operation.
HOLDA	5	OUT	Hold acknowledge. When active (high), HOLDA indicates that the processor is in the hold state and the address and data buses and memory control outputs (WE, MEMEN, and DBIN) are in the high-impedance state.
READY	62	IN	Ready. When active (high), READY indicates that memory will be ready to read or write during the next clock cycle. When not-ready is indicated during a memory operation, the TMS 9900 enters a wait state and suspends internal operation until the memory systems indicate ready.

FIGURE 2.9 (Continued)

within the TMS 9900, and then returning the data to the workspace. The TMS9900 memory map of the workspace is shown in Figure 2.10.

The first 32 words of memory (designated as memory address 0000 to 003E) are known as interrupt trap vectors. The TMS9900 has 16 interrupt levels, and the

TABLE 2 (CONCLUDED)

SIGNATURE	PIN	I/O	DESCRIPTION
WAIT	3	OUT	Wait. When active (high), WAIT indicates that the TMS 9900 has entered a wait state because of a not-ready condition from memory.
			TIMING AND CONTROL
IAQ	7	OUT	Instruction acquisition. IAQ is active (high) during any memory cycle when the TMS 9900 is acquiring an instruction. IAQ can be used to detect illegal op codes.
$\overline{\text{LOAD}}$	4	IN	Load. When active (low), $\overline{\text{LOAD}}$ causes the TMS 9900 to execute a nonmaskable interrupt with memory address $FFFC_{16}$ containing the trap vector (WP and PC). The load sequence begins after the instruction being executed is completed. $\overline{\text{LOAD}}$ will also terminate an idle state. If $\overline{\text{LOAD}}$ is active during the time $\overline{\text{RESET}}$ is released, then the $\overline{\text{LOAD}}$ trap will occur after the $\overline{\text{RESET}}$ function is completed. $\overline{\text{LOAD}}$ should remain active for one instruction period. IAQ can be used to determine instruction boundaries. This signal can be used to implement cold-start ROM loaders. Additionally, front-panel routines can be implemented using CRU bits as front-panel-interface signals and software-control routines to control the panel operations.
$\overline{\text{RESET}}$	6	IN	Reset. When active (low), $\overline{\text{RESET}}$ causes the processor to be reset and inhibits $\overline{\text{WE}}$ and CRUCLK. When $\overline{\text{RESET}}$ is released, the TMS 9900 then initiates a level-zero interrupt sequence that acquires WP and PC from locations 0000 and 0002, sets all status register bits to zero, and starts execution. $\overline{\text{RESET}}$ will also terminate an idle state. $\overline{\text{RESET}}$ must be held active for a minimum of three clock cycles.

FIGURE 2.9 (Continued)

workspace pointer (WP) and program counter (PC) corresponding to each interrupt level are stored in such memory locations. The next contiguous block of 32 words is used for the XOP (extended operation) instruction. The remaining memory is used for programs, data, and workspace registers, except for the last two memory words (FFFC and FFFE), which are used as the trap vector of the LOAD function.

The WP contains the memory address of the first word in the currently active set of workspace registers. The workspace register file consists of 16 contiguous memory words. Each register may function as an operand register, accumulator, address register, or index register. The processor executes instructions by adding the register number to the contents of the workspace pointer and initiating a memory request for that word.

The PC functions as in any other microprocessor and contains the address of the next instruction to be executed.

The SR contains flag bits that indicate results of the most recent arithmetic or logical operation performed. SR also contains a 4-bit interrupt mask.

The TMS9900 has seven basic addressing modes:

- Workspace register.
- Workspace register indirect.
- Workspace register indirect with auto increment.
- Indexed.
- Direct.
- Immediate.
- Relative.

In workspace register addressing the contents of the indicated workspace register R are the operand.

In workspace register indirect addressing, the contents of the indicated workspace register contain the memory address of the operand.

In workspace register indirect with auto increment addressing, the workspace register again contains the memory address of the operand. However, after the access the contents of the workspace register R are incremented by 1 for byte operations, or by 2 for word operations.

In indexed addressing, the word following the instruction contains the base address. The workspace register R contains the index value. The effective address of the operand is computed by adding the base address to the index value.

Indirect addressing, the word following the instruction contains the address of the operand.

In immediate addressing, the word following the instruction contains the operand.

In relative addressing, the 8-bit signed displacement in the right byte of the instruction is multiplied by 2 and added to the updated contents of the PC. The result is the next instruction for a JUMP type instruction. For a communications register unit (CRU) instruction, the 8-bit signed displacement is added to the CRU base address. The result is the CRU relative address required.

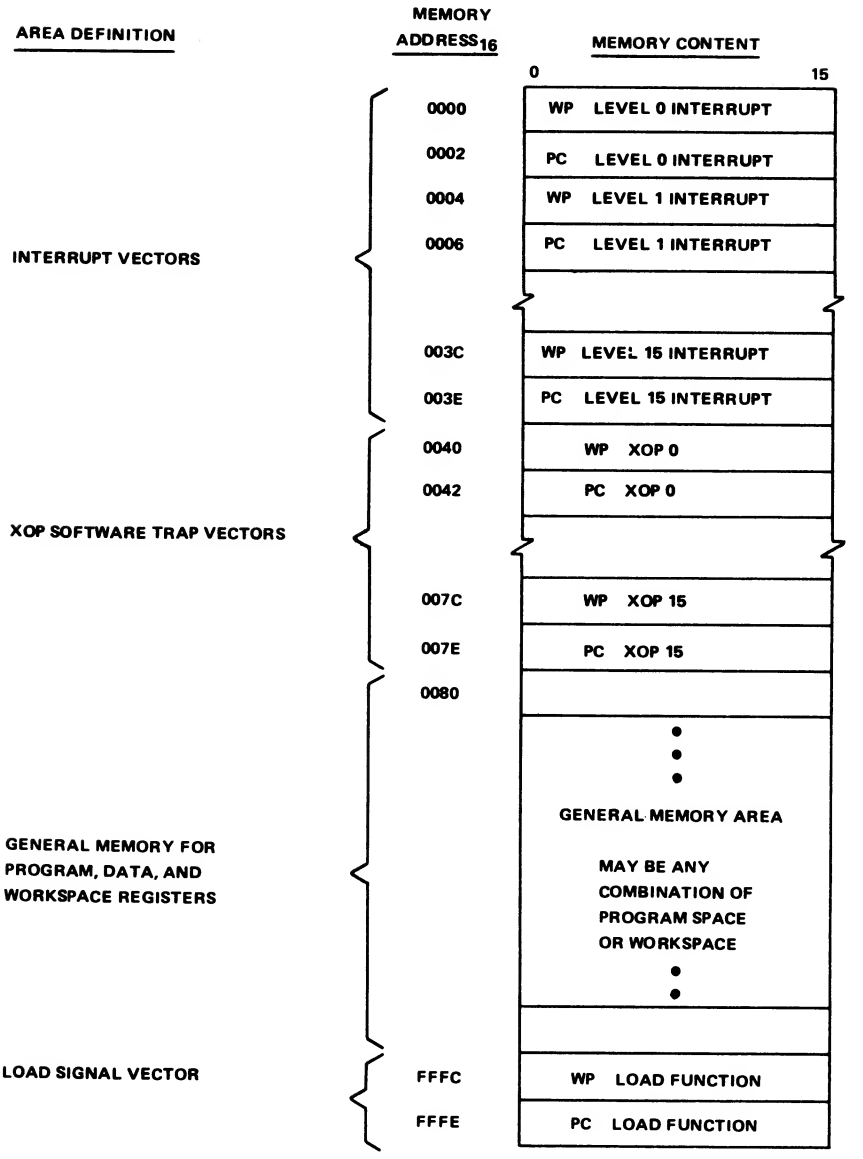


FIGURE 2.10 Texas Instruments TMS 9900 memory map. (Source: Texas Instruments Incorporated.)

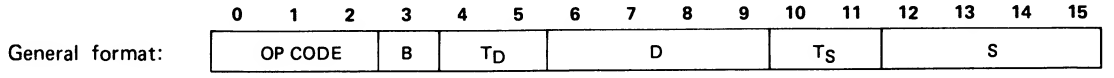
The TMS9900 instructions can be broken into 13 distinct groups:

- Dual operand instructions with multiple addressing modes for source and destination operand.
- Dual operand instructions with multiple addressing modes for the source operand and workspace register addressing for the destination.
- Extended operation.
- Single operand instructions.
- CRU multiple bit instructions.
- CRU single bit instructions.
- Jump instructions.
- Shift instructions.
- Immediate register instructions.
- Internal register load immediate instructions.
- Internal register store instructions.
- Return workspace pointer (RTWP) instruction.
- External instructions.

The TMS9900 instructions are presented in Table 2.3.

TABLE 2.3. TMS 9900 Instruction Set[a]

Dual Operand Instructions with Multiple Addressing Modes for Source and Destination Operand



If B = 1 the operands are bytes and the operand addresses are byte addresses. If B = 0 the operands are words and the operand addresses are word addresses.

The addressing mode for each operand is determined by the T field of that operand.

T_S OR T_D	S OR D	ADDRESSING MODE	NOTES
00	0, 1, . . . 15	Workspace register	1
01	0, 1, . . . 15	Workspace register indirect	
10	0	Symbolic	4
10	1, 2, . . . 15	Indexed	2,4
11	0, 1, . . . 15	Workspace register indirect auto-increment	3

NOTES: 1. When a workspace register is the operand of a byte instruction (bit 3 = 1), the left byte (bits 0 through 7) is the operand and the right byte (bits 8 through 15) is unchanged.
2. Workspace register 0 may not be used for indexing.
3. The workspace register is incremented by 1 for byte instructions (bit 3 = 1) and is incremented by 2 for word instructions (bit 3 = 0).
4. When T_S = T_D = 10, two words are required in addition to the instruction word. The first word is the source operand base address and the second word is the destination operand base address.

[a]*Reprinted by permission, copyright 1975, Texas Instrument Incorporated.*

Texas Instruments TMS9940

The TMS9940 is a 16-bit *n*-channel MOS microprocessor that was introduced in 1978. The TMS9940 is a member of the 9900 family of microprocessors and includes an on-chip memory of 128 bytes of RAM and 2K bytes of mask-programmable or reprogrammable ROM.

The functional block diagram of the TMS9940 is shown in Figure 2.11. The unit features three temporary registers, labeled T_1, T_2, and T_3, which store intermediate addresses and operands. The program counter (PC) and workspace pointer (WP) are similar in function to the TMS9900. The 9940 includes three separate I/O interfaces: a general-purpose on-chip I/O, an I/O expansion, and a multiprocessor system interface (MPSI).

The general-purpose on-chip I/O consists of 32 lines that may act either as input or output ports. Like the 9900, the 9940 I/O interface operates through a communications register unit (CRU), which includes three dedicated I/O pins (CRU-IN, CRU-OUT, and CRU-CLOCK) and instructions that drive the CRU to set, reset, or test any bit in the CRU array or move data between memory and I/O.

Another I/O configuration is a direct expandability to as many as 256 bits of additional I/O lines through the use of external logic. Figure 2.12 shows the I/O expansion configuration.

The last I/O interface is the MPSI, which is shown in the lower right hand section of the block diagram in Figure 2.11. The MPSI is a CRU compatible two-wire interface for transferring data in a multiprocessor configuration.

The multiprocessing configuration makes use of the T_C and T_D ports on the 9940. The basic configuration is shown in Figure 2.13, and can be extended for more than two processors.

In the multiprocessing configuration, the T_D signal is used to feed an internal 16-bit shift register, and the T_C input signal is used to clock the shift register to allow the data to be shifted independently of the rest of the CPU's operation. The processor sending the data is designated the "sender," and the processor receiving the data is designated the "receiver." The sender transmits an interrupt to the receiver, and the receiver's interrupt routine contains an STCR instruction, which stores the CRU data in memory, to receive the data from the sender. The complete software routine for transferring data from sender to receiver is described below. Reference can be made to the instruction set of the TMS9900, since the 9940 instruction set is a subset thereof.

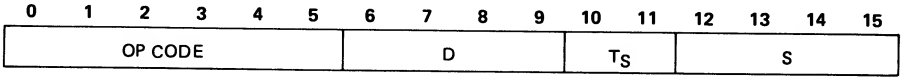
The sender loads the memory data to be transmitted to the receiver on the CRU by use of the LDCR instruction. The T_D signal then sends the data, and the T_C sig-

TABLE 2.3. (*Continued*)

MNEMONIC	OP CODE 0 1 2	B 3	MEANING	RESULT COMPARED TO 0	STATUS BITS AFFECTED	DESCRIPTION
A	1 0 1	0	Add	Yes	0-4	(SA)+(DA) → (DA)
AB	1 0 1	1	Add bytes	Yes	0-5	(SA)+(DA) → (DA)
C	1 0 0	0	Compare	Yes	0-2	Compare (SA) to (DA) and set appropriate status bits
CB	1 0 0	1	Compare bytes	Yes	0-2,5	Compare (SA) to (DA) and set appropriate status bits
S	0 1 1	0	Subtract	Yes	0-4	(DA) – (SA) → (DA)
SB	0 1 1	1	Subtract bytes	Yes	0-5	(DA) – (SA) → (DA)
SOC	1 1 1	0	Set ones corresponding	Yes	0-2	(DA) OR (SA) → (DA)
SOCB	1 1 1	1	Set ones corresponding bytes	Yes	0-2,5	(DA) OR (SA) → (DA)
SZC	0 1 0	0	Set zeroes corresponding	Yes	0-2	(DA) AND ($\overline{SA}$) → (DA)
SZCB	0 1 0	1	Set zeroes corresponding bytes	Yes	0-2,5	(DA) AND ($\overline{SA}$) → (DA)
MOV	1 1 0	0	Move	Yes	0-2	(SA) → (DA)
MOVB	1 1 0	1	Move bytes	Yes	0-2,5	(SA) → (DA)

Dual Operand Instructions with Multiple Addressing Modes for the Source Operand and Workspace Register Addressing for the Destination

General format:



The addressing mode for the source operand is determined by the T_S field.

T_S	S	ADDRESSING MODE	NOTES
00	0, 1, . . . 15	Workspace register	
01	0, 1, . . . 15	Workspace register indirect	
10	0	Symbolic	
10	1, 2, . . . 15	Indexed	1
11	0, 1, . . . 15	Workspace register indirect auto increment	2

NOTES: 1. Workspace register 0 may not be used for indexing.
2. The workspace register is incremented by 2.

MNEMONIC	OP CODE 0 1 2 3 4 5	MEANING	RESULT COMPARED TO 0	STATUS BITS AFFECTED	DESCRIPTION
COC	0 0 1 0 0 0	Compare ones corresponding	No	2	Test (D) to determine if 1's are in each bit position where 1's are in (SA). If so, set ST2.
CZC	0 0 1 0 0 1	Compare zeros corresponding	No	2	Test (D) to determine if 0's are in each bit position where 1's are in (SA). If so, set ST2.
XOR	0 0 1 0 1 0	Exclusive OR	Yes	0-2	(D) ⊕ (SA) → (D)
MPY	0 0 1 1 1 0	Multiply	No		Multiply unsigned (D) by unsigned (SA) and place unsigned 32-bit product in D (most significant) and D+1 (least significant). If WR15 is D, the next word in memory after WR15 will be used for the least significant half of the product.
DIV	0 0 1 1 1 1	Divide	No	4	If unsigned (SA) is less than unsigned (D), perform no operation and set ST4. Otherwise, divide unsigned (D) and (D+1) by unsigned (SA). Quotient → (D), remainder → (D+1). If D = 15, the next word in memory after WR15 will be used for the remainder.

TABLE 2.3. (*Continued*)

Extended Operation (XOP) Instruction

General format:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	0	1	1	D				T_S		S			

The T_S and S fields provide multiple mode addressing capability for the source operand. When the XOP is executed, ST6 is set and the following transfers occur:

$(40_{16} + 4D) \rightarrow$ (WP)
$(41_{16} + 4D) \rightarrow$ (PC)
SA $\rightarrow$ (new WR11)
(old WP) $\rightarrow$ (new WR13)
(old PC) $\rightarrow$ (new WR14)
(old ST) $\rightarrow$ (new WR15)

The TMS 9900 does not test interrupt requests ($\overline{\text{INTREQ}}$) upon completion of the XOP instruction.

Single Operand Instructions

General format:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
OP CODE										T_S		S			

The T_S and S fields provide multiple mode addressing capability for the source operand.

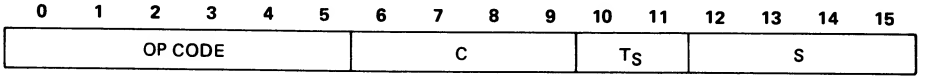
MNEMONIC	OP CODE 0 1 2 3 4 5 6 7 8 9	MEANING	RESULT COMPARED TO 0	STATUS BITS AFFECTED	DESCRIPTION
B	0 0 0 0 0 1 0 0 0 1	Branch	No	–	(SA) → (PC)
BL	0 0 0 0 0 1 1 0 1 0	Branch and link	No	–	(PC) → (WR11); SA → (PC)
BLWP	0 0 0 0 0 1 0 0 0 0	Branch and load workspace pointer	No	–	(SA) → (WP);(SA+1) → (PC); (old WP) → (new WR13); (old PC) → (new WR14); (old ST) → (new WR15); the interrupt input ($\overline{\text{INTREQ}}$) is not tested upon completion of the BLWP instruction.
CLR	0 0 0 0 0 1 0 0 1 1	Clear operand	No	–	0 → (SA)
SETO	0 0 0 0 0 1 1 1 0 0	Set to ones	No	–	$FFFF_{16}$ → (SA)
INV	0 0 0 0 0 1 0 1 0 1	Invert	Yes	0-2	$(\overline{SA})$ → (SA)
NEG	0 0 0 0 0 1 0 1 0 0	Negate	Yes	0-4	–(SA) → (SA)
ABS	0 0 0 0 0 1 1 1 0 1	Absolute value	No	0-4	|(SA)| → (SA)
SWPB	0 0 0 0 0 1 1 0 1 1	Swap bytes	No	–	(SA), bits 0 thru 7 → (SA), bits 8 thru 15; (SA), bits 8 thru 15 → (SA), bits 0 thru 7.
INC	0 0 0 0 0 1 0 1 1 0	Increment	Yes	0-4	(SA) + 1 → (SA)
INCT	0 0 0 0 0 1 0 1 1 1	Increment by two	Yes	0-4	(SA) + 2 → (SA)
DEC	0 0 0 0 0 1 1 0 0 0	Decrement	Yes	0-4	(SA) – 1 → (SA)
DECT	0 0 0 0 0 1 1 0 0 1	Decrement by two	Yes	0-4	(SA) – 2 → (SA)
X	0 0 0 0 0 1 0 0 1 0	Execute	No	–	Execute the instruction at SA. (see note)

NOTE: If additional memory words for the execute instruction are required to define the operands of the instruction located at SA, these words will be accessed from PC and the PC will be updated accordingly. The instruction acquisition signal (IAQ) will not be true when the TMS 9900 accesses the instruction at SA. Status bits are affected in the normal manner for the instruction executed.

TABLE 2.3. (*Continued*)

CRU Multiple-Bit Instructions

General format:



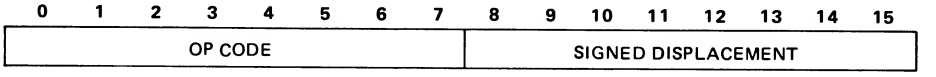
The C field specifies the number of bits to be transferred. If C = 0, 16 bits will be transferred. The CRU base register (WR12, bits 3 through 14) defines the starting CRU bit address. The bits are transferred serially and the CRU address is incremented with each bit transfer, although the contents of WR12 is not affected. T_S and S provide multiple mode addressing capability for the source operand. If 8 or fewer bits are transferred (C = 1 through 8), the source address is a byte address. If 9 or more bits are transferred (C = 0, 9 through 15), the source address is a word address. If the source is addressed in the workspace register indirect auto increment mode, the workspace register is incremented by 1 if C = 1 through 8, and is incremented by 2 otherwise.

MNEMONIC	OP CODE 0 1 2 3 4 5	MEANING	RESULT COMPARED TO 0	STATUS BITS AFFECTED	DESCRIPTION
LDCR	0 0 1 1 0 0	Load communcation register	Yes	0-2,5†	Beginning with LSB of (SA), transfer the specified number of bits from (SA) to the CRU.
STCR	0 0 1 1 0 1	Store communcation register	Yes	0-2,5†	Beginning with LSB of (SA), transfer the specified number of bits from the CRU to (SA). Load unfilled bit positions with 0.

†ST5 is affected only if $1 \leqslant C \leqslant 8$.

CRU Single-Bit Instructions

General format:

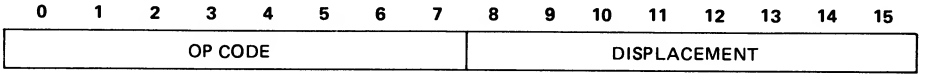


CRU relative addressing is used to address the selected CRU bit.

MNEMONIC	OP CODE 0 1 2 3 4 5 6 7	MEANING	STATUS BITS AFFECTED	DESCRIPTION
SBO	0 0 0 1 1 1 0 1	Set bit to one	–	Set the selected CRU output bit to 1.
SBZ	0 0 0 1 1 1 1 0	Set bit to zero	–	Set the selected CRU output bit to 0.
TB	0 0 0 1 1 1 1 1	Test bit	2	If the selected CRU input bit = 1, set ST2.

Jump Instructions

General format:



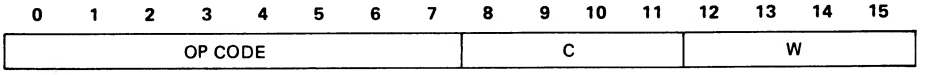
Jump instructions cause the PC to be loaded with the value selected by PC relative addressing if the bits of ST are at specified values. Otherwise, no operation occurs and the next instruction is executed since PC points to the next instruction. The displacement field is a word count to be added to PC. Thus, the jump instruction has a range of –128 to 127 words from memory-word address following the jump instruction. No ST bits are affected by jump instruction.

TABLE 2.3. (*Continued*)

MNEMONIC	OP CODE 0 1 2 3 4 5 6 7	MEANING	ST CONDITION TO LOAD PC
JEQ	0 0 0 1 0 0 1 1	Jump equal	ST2 = 1
JGT	0 0 0 1 0 1 0 1	Jump greater than	ST1 = 1
JH	0 0 0 1 1 0 1 1	Jump high	ST0 = 1 and ST2 = 0
JHE	0 0 0 1 0 1 0 0	Jump high or equal	ST0 = 1 or ST2 = 1
JL	0 0 0 1 1 0 1 0	Jump low	ST0 = 0 and ST2 = 0
JLE	0 0 0 1 0 0 1 0	Jump low or equal	ST0 = 0 or ST2 = 1
JLT	0 0 0 1 0 0 0 1	Jump less than	ST1 = 0 and ST2 = 0
JMP	0 0 0 1 0 0 0 0	Jump unconditional	unconditional
JNC	0 0 0 1 0 1 1 1	Jump no carry	ST3 = 0
JNE	0 0 0 1 0 1 1 0	Jump not equal	ST2 = 0
JNO	0 0 0 1 1 0 0 1	Jump no overflow	ST4 = 0
JOC	0 0 0 1 1 0 0 0	Jump on carry	ST3 = 1
JOP	0 0 0 1 1 1 0 0	Jump odd parity	ST5 = 1

Shift Instructions

General format:

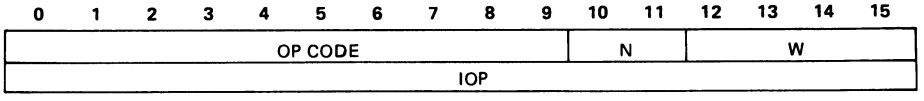


If C = 0, bits 12 through 15 of WR0 contain the shift count. If C = 0 and bits 12 through 15 of WR0 = 0, the shift count is 16.

MNEMONIC	OP CODE 0 1 2 3 4 5 6 7	MEANING	RESULT COMPARED TO 0	STATUS BITS AFFECTED	DESCRIPTION
SLA	0 0 0 0 1 0 1 0	Shift left arithmetic	Yes	0-4	Shift (W) left. Fill vacated bit positions with 0.
SRA	0 0 0 0 1 0 0 0	Shift right arithmetic	Yes	0-3	Shift (W) right. Fill vacated bit positions with original MSB of (W).
SRC	0 0 0 0 1 0 1 1	Shift right circular	Yes	0-3	Shift (W) right. Shift previous LSB into MSB.
SRL	0 0 0 0 1 0 0 1	Shift right logical	Yes	0-3	Shift (W) right. Fill vacated bit positions with 0's.

Immediate Register Instructions

General format:

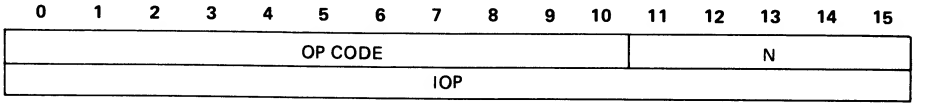


MNEMONIC	OP CODE 0 1 2 3 4 5 6 7 8 9 10	MEANING	RESULT COMPARED TO 0	STATUS BITS AFFECTED	DESCRIPTION
AI	0 0 0 0 0 0 1 0 0 0 1	Add immediate	Yes	0-4	(W) + IOP → (W)
ANDI	0 0 0 0 0 0 1 0 0 1 0	AND immediate	Yes	0-2	(W) AND IOP → (W)
CI	0 0 0 0 0 0 1 0 1 0 0	Compare immediate	Yes	0-2	Compare (W) to IOP and set appropriate status bits
LI	0 0 0 0 0 0 1 0 0 0 0	Load immediate	Yes	0-2	IOP → (W)
ORI	0 0 0 0 0 0 1 0 0 1 1	OR immediate	Yes	0-2	(W) OR IOP → (W)

TABLE 2.3. (*Continued*)

Internal Register Load Immediate Instructions

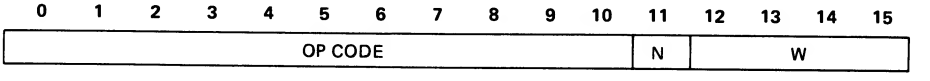
General format:



MNEMONIC	OP CODE 0 1 2 3 4 5 6 7 8 9 10	MEANING	DESCRIPTION
LWPI	0 0 0 0 0 0 1 0 1 1 1	Load workspace pointer immediate	IOP → (WP), no ST bits affected
LIMI	0 0 0 0 0 0 1 1 0 0 0	Load interrupt mask	IOP, bits 12 thru 15 → ST12 thru ST15

Internal Register Store Instructions

General format:

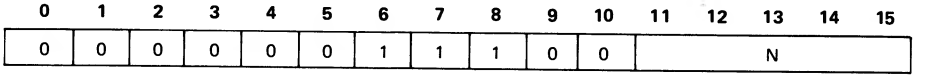


No ST bits are affected.

MNEMONIC	OP CODE 0 1 2 3 4 5 6 7 8 9 10	MEANING	DESCRIPTION
STST	0 0 0 0 0 0 1 0 1 1 0	Store status register	(ST) → (W)
STWP	0 0 0 0 0 0 1 0 1 0 1	Store workspace pointer	(WP) → (W)

Return Workspace Pointer (RTWP) Instruction

General format:

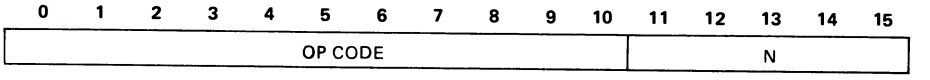


The RTWP instruction causes the following transfers to occur:

(WR15) → (ST)
(WR14) → (PC)
(WR13) → (WP)

External Instructions

General format:



External instructions cause the three most-significant address lines (A0 through A2) to be set to the below-described levels and the CRUCLK line to be pulsed, allowing external control functions to be initiated.

MNEMONIC	OP CODE 0 1 2 3 4 5 6 7 8 9 10	MEANING	STATUS BITS AFFECTED	DESCRIPTION	ADDRESS BUS A0 A1 A2
IDLE	0 0 0 0 0 0 1 1 0 1 0	Idle	–	Suspend TMS 9900 instruction execution until an interrupt, $\overline{\text{LOAD}}$, or $\overline{\text{RESET}}$ occurs	L H L
RSET	0 0 0 0 0 0 1 1 0 1 1	Reset	12–15	0 → ST12 thru ST15	L H H
CKOF	0 0 0 0 0 0 1 1 1 1 0	User defined		———	H L H
CKON	0 0 0 0 0 0 1 1 1 0 1	User defined		———	H H L
LREX	0 0 0 0 0 0 1 1 1 1 1	User defined		———	H H H

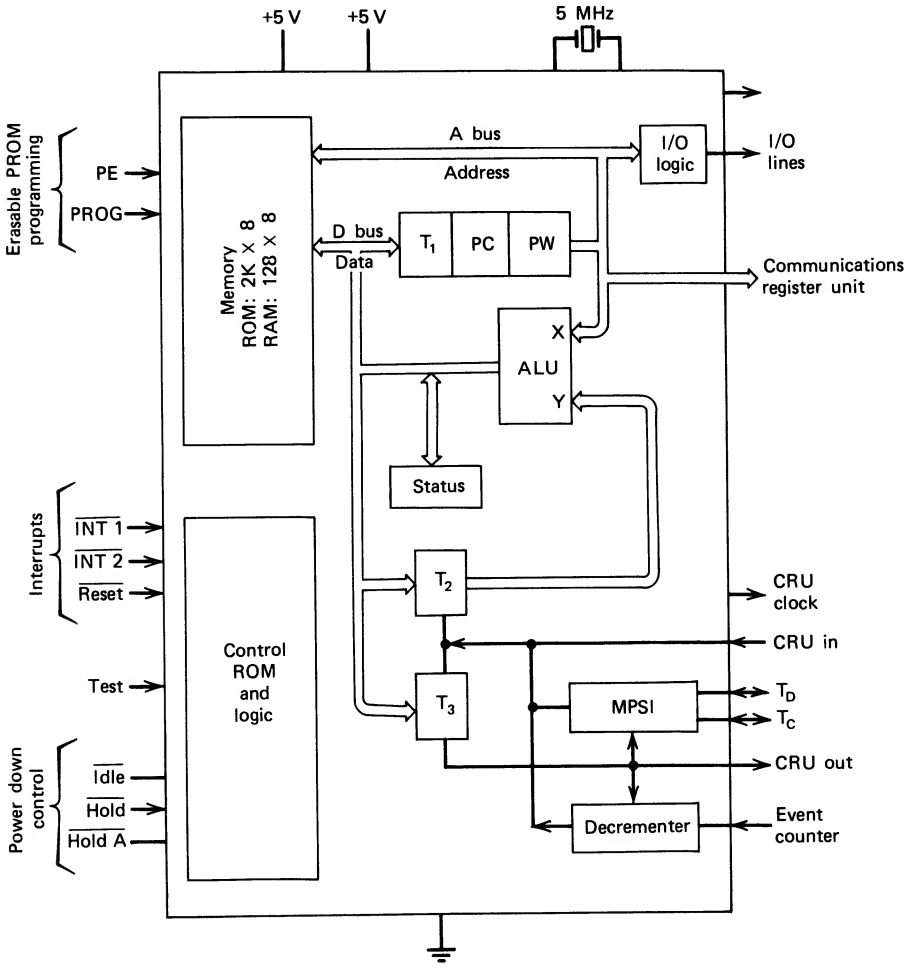


FIGURE 2.11 Texas Instruments TMS 9940 functional block diagram. (Source: Texas Instruments Incorporated.)

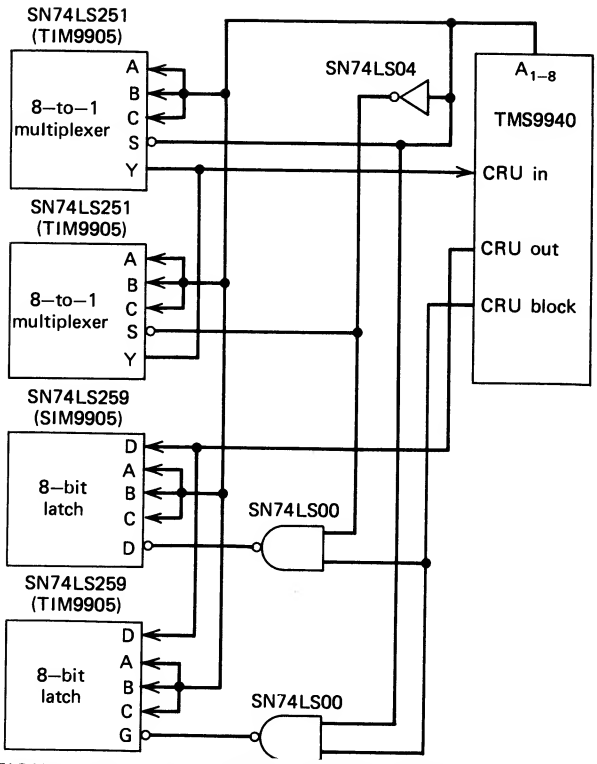


FIGURE 2.12 TMS 9940 I/O expansion. (Source: Texas Instruments Incorporated.)

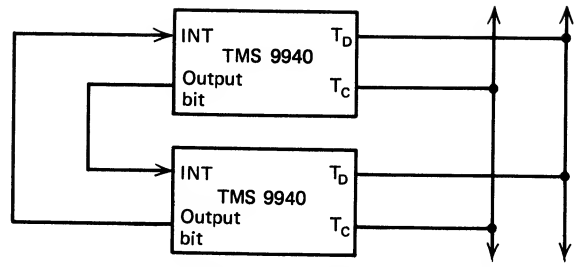


FIGURE 2.13 TMS 9940 multiprocessor configuration. (Source: Texas Instruments Incorporated.)

nal sends out the clock strobe. The receiver accepts the data by sending an acknowledgment back and checks whether the transmission is complete. If not, it enters a wait loop until the next data word is sent. When the transfer is complete, the receiver returns to its original program.

Fairchild 9440 Microflame

The Fairchild 9440 Microflame is a single-chip 16-bit bipolar microprocessor implemented using Fairchild's Isoplanar integrated injection logic technology (I^3L™). The 9440 executes the same instruction set as the Data General NOVA line of minicomputers.

The functional block diagram of the 9440 is shown in Figure 2.14, and the pin diagram is shown in Figure 2.15. Some of the principal features of the 9440 as shown in the block diagram are:

- A microprogram programmed logic array (PLA).
- Four 16-bit accumulators.
- A 4-bit ALU that operates on two 16-bit words in four consecutive steps.
- Four special-purpose 16-bit registers: scratch register, bus register, instruction register, and program counter.
- Fifteen-bit address lines, which give the 9440 a direct memory addressing capability of 32,768 16-bit words.

A typical system using the 9440 is shown in Figure 2.16. The support circuits include the 9441 memory control unit, the 9442 I/O control unit, and either the 93481 4K I^3L bipolar dynamic memory or the 93483 16K I^3L bipolar dynamic memory.

The 9440 has four basic addressing modes: absolute addressing of page zero (i.e., the first 256 memory locations), and relative addressing using either of two accumulators or the program counter. Since each of these four basic modes may be either direct or indirect, the total number of addressing modes is 8.

The instruction set of the 9440 is divided into three groups:

- Memory reference instructions.
- Arithmetic/logic instructions.
- Input/output instructions.

The list of 9440 instructions follow, with reference to Figures 2.17, 2.18, and 2.19 to illustrate the fields of the various instructions.

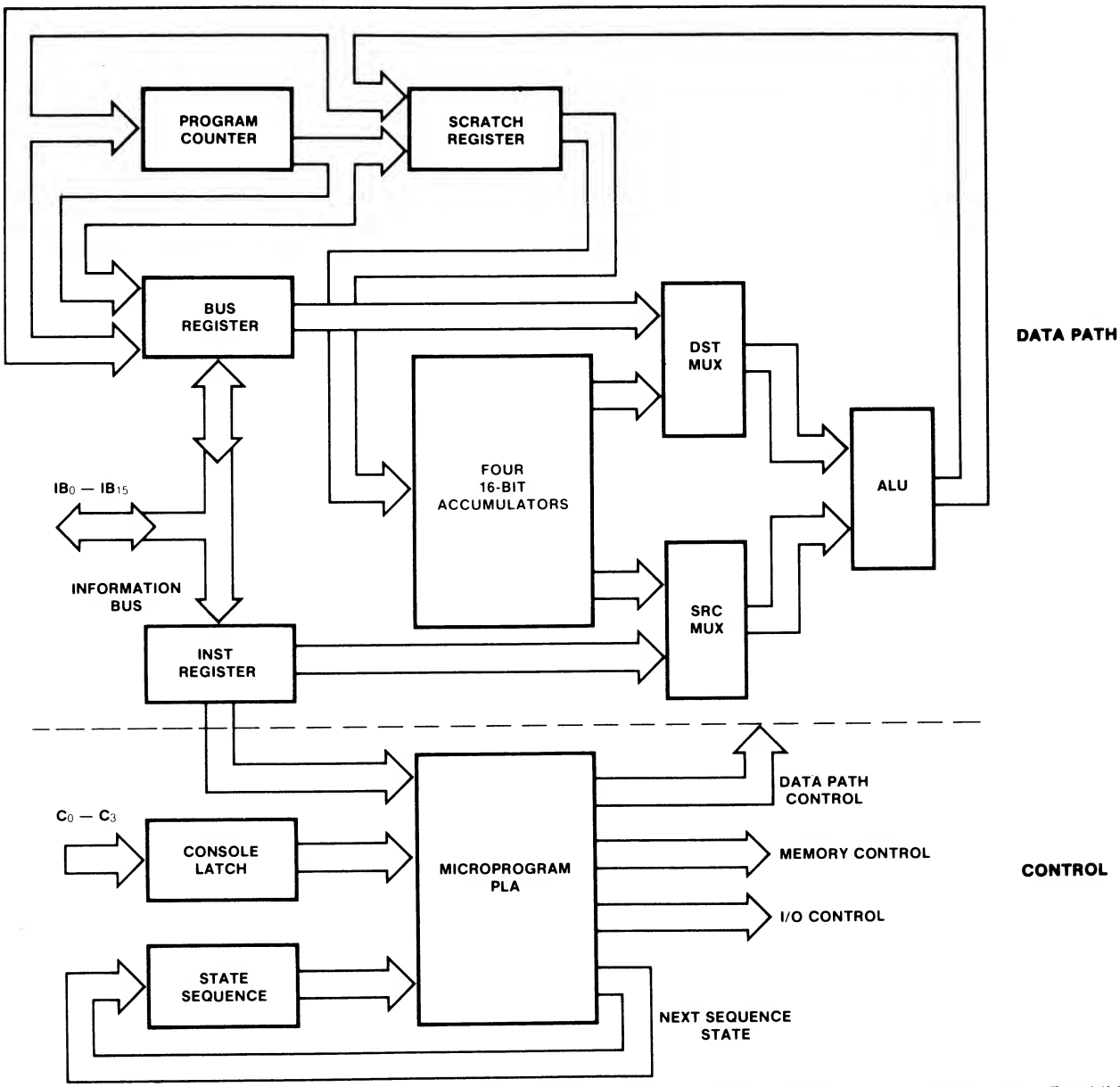


FIGURE 2.14 Fairchild 9440 Microflame functional block diagram. (Source: Fairchild. Reprinted by permission from Fairchild Camera and Instrument Corporation.)

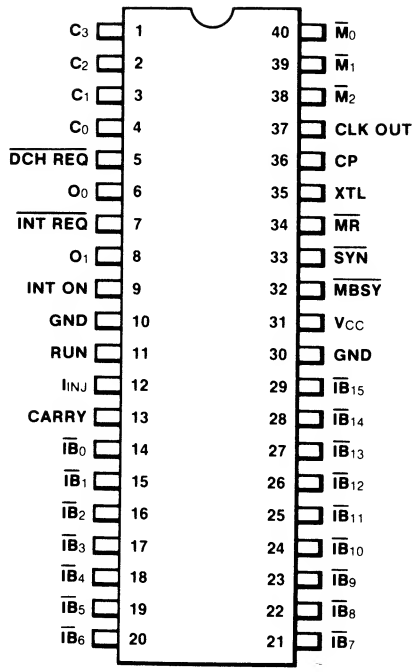


FIGURE 2.15 Fairchild 9440 pin diagram. (Source: Fairchild. Reprinted by permission from Fairchild Camera and Instrument Corporation.)

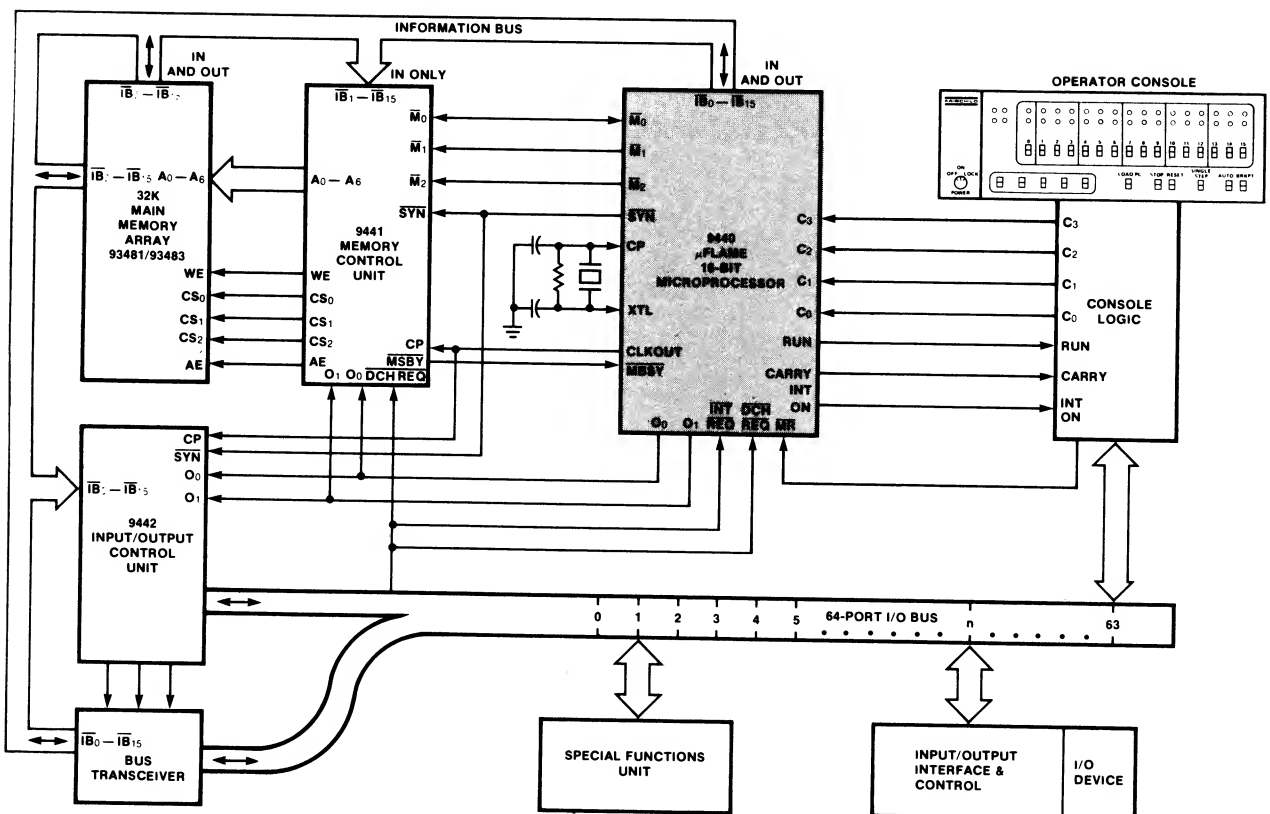


FIGURE 2.16 Fairchild 9440 typical system diagram. (Source: Fairchild. Reprinted by permission from Fairchild Camera and Instrument Corporation.)

Fairchild 9440 Microflame Instruction Set

The Fairchild 9445 is another version of the Fairchild 9440 microprocessor capable of signed on-chip multiply and divide operations.

9440 INSTRUCTIONS

Memory Reference instructions without register are used for branching (JMP, JSR) without involving accumulators. These instructions are also used for modifying memory (ISZ, DSZ). Memory Reference instructions with register are used to move 16-bit words between the memory and the accumulators.

JMP — *Jump.* The 9440 loads the effective address into the PC. The program sequence continues from that location.

JSR — *Jump to SubRoutine.* The 9440 calculates the effective address, then stores PC + 1 into AC3, loads the effective address into the PC and continues instruction execution from that location.

ISZ — *Increment and Skip if Zero.* The 9440 reads a word from the memory location specified by the effective address, increments the word and writes the result back into the same location. If the incremented word is zero, the 9440 skips the next instruction.

DSZ — *Decrement and Skip if Zero.* The 9440 reads a word from the memory location specified by the effective address; decrements it and writes the result back into the same location. If the result is zero the 9440 skips the next instruction.

LDA — *Load Accumulator.* The 9440 loads the contents of the memory location specified by the effective address into the accumulator defined by bits 3 and 4 of the instruction.

STA — *Store Accumulator.* The 9440 stores the contents of the selected accumulator into the memory location specified by the effective address; the accumulator contents are not affected.

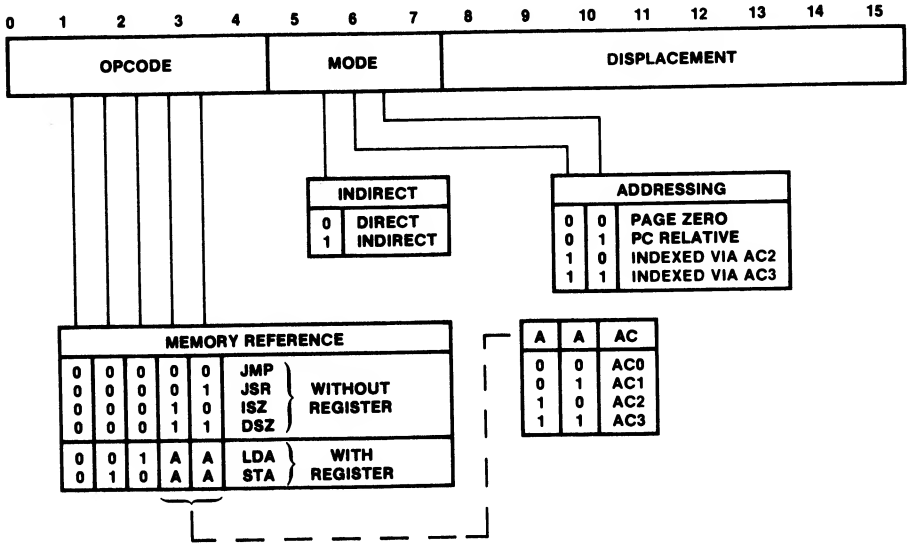
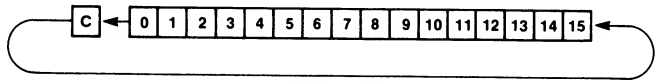
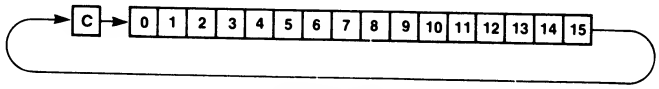


FIGURE 2.17 Fairchild 9440 memory reference instructions. (Source: Fairchild. Reprinted by permission from Fairchild Camera and Instrument Corporation.)

Arithmetic/Logic instructions perform arithmetic (ADD, ADC, INC, NEG, SUB) or Boolean (AND, COM, MOV) operations on the contents of two registers. The results of each operation together with the carry bit form a 17-bit word that can be rotated once to the left or to the right.



Rotate Left



Rotate Right

The carry bit is calculated according to a base value, as specified by the carry code. That base value is complemented if the arithmetic operation produces a carry out of bit 0; otherwise the carry bit is equal to the base value. During a byte swap operation, bits 0-7 and 8-15 of the result are swapped. The carry bit is not affected.

After the shift and still as part of the same arithmetic/logic instruction, the 9440 checks the shifted word and the new carry bit for zero or non-zero results. According to the skip code, the 9440 may skip the next instruction.

The 17-bit shifted word is loaded into the Carry FF and the destination accumulator only if bit 12 of the instruction is '0'. Loading is inhibited if bit 12 is '1'.

Arithmetic and Logic Functions

COM — *Complement.* The 9440 operates on the contents of the source accumulator, logically complementing it.

NEG — *Negate.* The 9440 calculates the 2's complement of the contents of the source accumulator. The base value for the carry bit is complemented only if SRC contained '0'.

MOV — *Move.* The 9440 takes the contents of the source accumulator, together with the base value for the carry bit, and performs the operation specified by the shift code as explained above.

INC — *Increment.* The 9440 adds '1' to the contents of the source accumulator.

ADC — *Add Complement.* The 9440 complements the contents of the source accumulator and adds it to the contents of the destination accumulator.

SUB — *Subtract.* The 9440 calculates the 2's complement of the contents of the source accumumlator and adds it to the contents of the destination accumulator (DST - SRC).

ADD — *Add.* The 9440 adds the contents of the source accumulator and the contents of the destination accumulator (SRC + DST).

AND — *And.* The 9440 computes the logic AND function on the source and destination accumulators (SRC•DST).

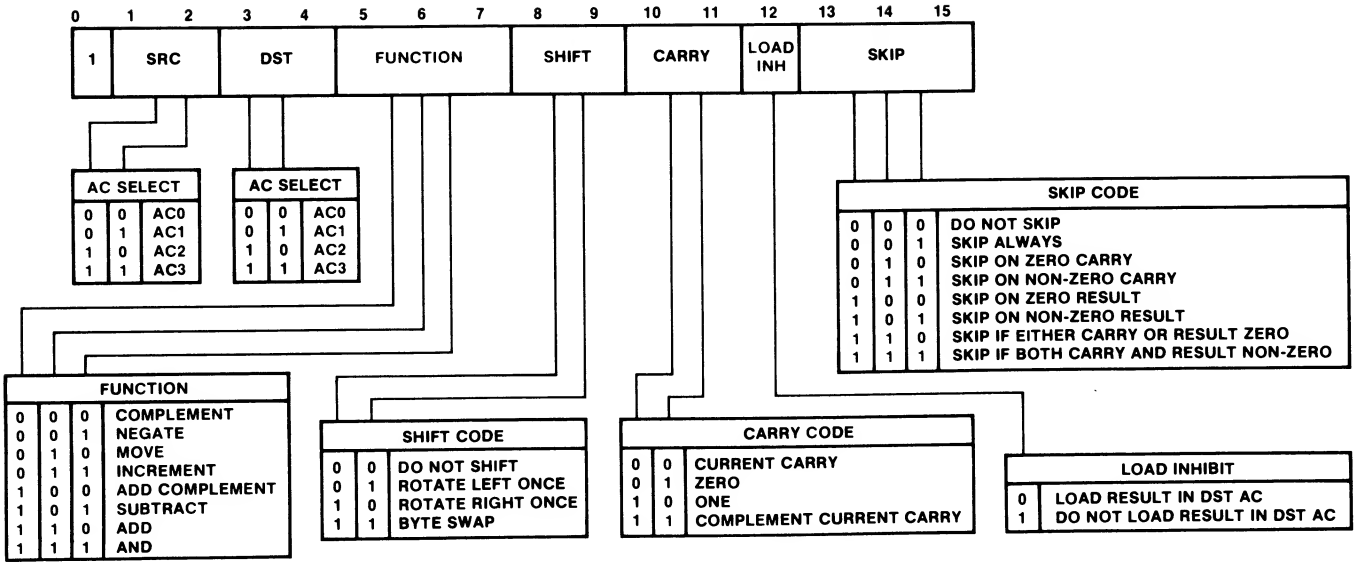

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
1
SRC
DST
FUNCTION
SHIFT
CARRY
LOAD INH
SKIP
AC SELECT
0 0 AC0
0 1 AC1
1 0 AC2
1 1 AC3
AC SELECT
0 0 AC0
0 1 AC1
1 0 AC2
1 1 AC3
SKIP CODE
0 0 0 DO NOT SKIP
0 0 1 SKIP ALWAYS
0 1 0 SKIP ON ZERO CARRY
0 1 1 SKIP ON NON-ZERO CARRY
1 0 0 SKIP ON ZERO RESULT
1 0 1 SKIP ON NON-ZERO RESULT
1 1 0 SKIP IF EITHER CARRY OR RESULT ZERO
1 1 1 SKIP IF BOTH CARRY AND RESULT NON-ZERO
FUNCTION
0 0 0 COMPLEMENT
0 0 1 NEGATE
0 1 0 MOVE
0 1 1 INCREMENT
1 0 0 ADD COMPLEMENT
1 0 1 SUBTRACT
1 1 0 ADD
1 1 1 AND
SHIFT CODE
0 0 DO NOT SHIFT
0 1 ROTATE LEFT ONCE
1 0 ROTATE RIGHT ONCE
1 1 BYTE SWAP
CARRY CODE
0 0 CURRENT CARRY
0 1 ZERO
1 0 ONE
1 1 COMPLEMENT CURRENT CARRY
LOAD INHIBIT
0 LOAD RESULT IN DST AC
1 DO NOT LOAD RESULT IN DST AC

FIGURE 2.18 Fairchild 9440 arithmetic/logic instructions. (Source: Fairchild. Reprinted by permission from Fairchild Camera and Instrument Corporation.)

Input/Output instructions move data between the 9440 accumulators and three buffers in the peripheral device interface. These instructions also perform control functions in the I/O device and test the status flags in both the peripheral circuitry and the central processor.

NIO — *No Input/Output transfer.* The I/O device specified by the device code performs the control function defined by bits 8 and 9 of the instruction.

DIA/B/C — *Data In A or B or C.* The I/O device specified by the device code performs the control function and sends the contents of its A, B or C buffers to the 9440. The processor loads that data into the selected accumulator.

DOA/B/C — *Data Out A or B or C.* The 9440 sends the contents of the selected accumulator to the I/O device specified by the device code. The I/O device stores the data in its A, B or C buffers and performs the control function.

SKPBN — *Skip if Busy is Non-Zero.* The 9440 skips the next instruction if the busy flag in the I/O device specified by the device code is '1'. No control function is executed.

SKPBZ — *Skip if Busy is Zero.* The 9440 skips the next instruction if the busy flag in the I/O devices specified by the device code is '0'. No control function is executed.

SKPDN — *Skip if Done is Non-Zero.* The 9440 skips the next instruction if the done flag in the I/O device specified by the device code is '1'. No control function is executed.

SKPDZ — *Skip if Done is Zero.* The 9440 skips the next instruction if the done flag in the I/O device specified by the device code is '0'. No control function is executed.

Special Device Code-77 Instruction — Device code 77 (octal) is used for some special instructions. Some like IORST or Mask Out are common to all I/O devices; others, like Interrupt Enable or Halt are special CPU instructions. In all device code-77 instructions except the skip instructions, bits 8 and 9 control the Interrupt Enable flip-flop in the 9440; An '01' code is used to set it, '10' to reset it.

NIOS CPU — *Interrupt Enable.* The 9440 sets the Interrupt-Enable flip-flop after a delay of one instruction to allow for jumping back from an interrupt service routine.

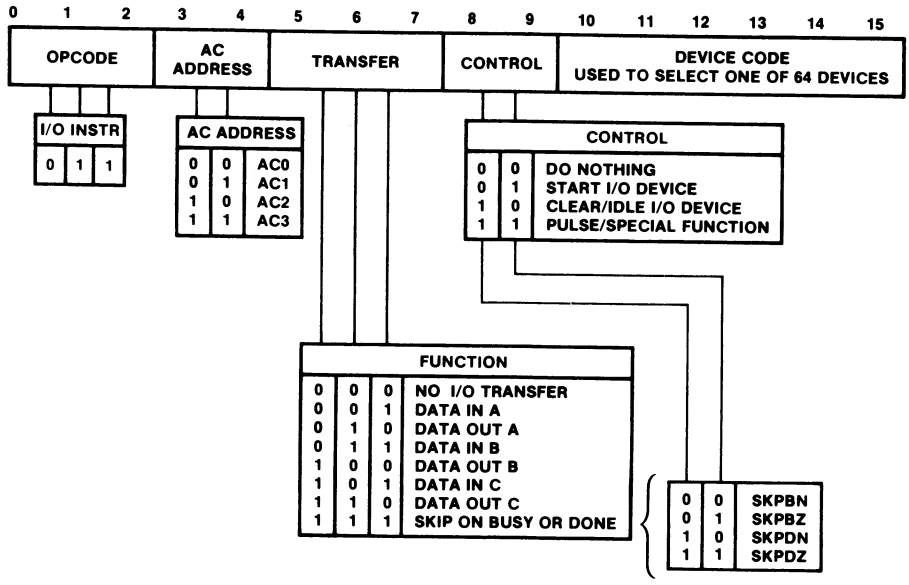

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
OPCODE
AC ADDRESS
TRANSFER
CONTROL
DEVICE CODE USED TO SELECT ONE OF 64 DEVICES
I/O INSTR
0 1 1
AC ADDRESS
0 0 AC0
0 1 AC1
1 0 AC2
1 1 AC3
CONTROL
0 0 DO NOTHING
0 1 START I/O DEVICE
1 0 CLEAR/IDLE I/O DEVICE
1 1 PULSE/SPECIAL FUNCTION
FUNCTION
0 0 0 NO I/O TRANSFER
0 0 1 DATA IN A
0 1 0 DATA OUT A
0 1 1 DATA IN B
1 0 0 DATA OUT B
1 0 1 DATA IN C
1 1 0 DATA OUT C
1 1 1 SKIP ON BUSY OR DONE
0 0 SKPBN
0 1 SKPBZ
1 0 SKPDN
1 1 SKPDZ

FIGURE 2.19 Fairchild 9440 input/output instructions. (Source: Fairchild. Reprinted by permission from Fairchild Camera and Instrument Corporation.)

Motorola 68000

The Motorola 68000 is a 16-bit *n*-channel MOS microprocessor fabricated using Motorola's scaled down HMOS processing and packaged in a 64-pin package. The 6800 was announced in 1979.

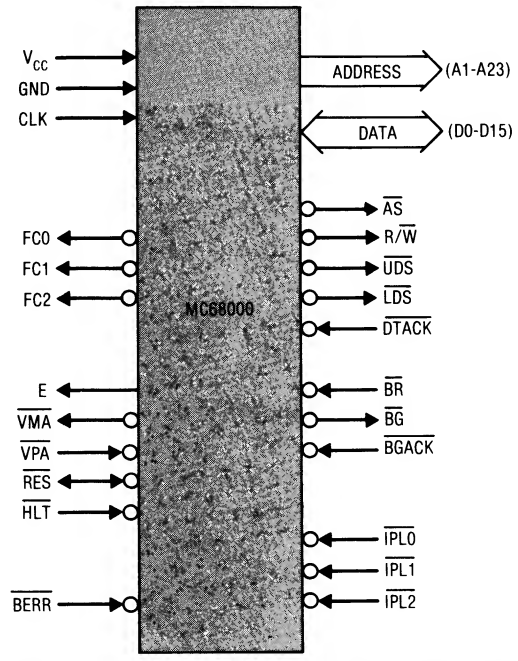
The pin diagram and definition of the signal lines of the 68000 are given in Figure 2.20. As the figure indicates, the 68000 has 24 address lines and therefore is capable of directly addressing 8 megabytes of memory.

The architecture of the 68000 can best be described with reference to the various registers in the device. The general-purpose registers are shown in Figure 2.21, and the status register is shown in Figure 2.22.

The general-purpose registers are divided into two groups:

- Eight data registers.
- Eight address registers.

The adjectives "data" and "address" describe the principal use of such registers. The data registers are intended to store data and to be used for manipulating data. They may be used as sources and destinations for certain instructions, but are only used in addressing as index registers. The address registers, as the name implies, are for addressing. The stack pointer is one of the address registers.



A1-A23	Address leads	23-bit address bus; capable of addressing 16,777,216 bytes in conjunction with UDS and LDS.
D0-D15	Data leads	16-bit data bus; transfers 8 or 16 bits of information.
AS	Address strobe	Indicates valid address and provides a bus lock for indivisible operations.
R/W	Read/write	Defines bus operation as read or write and controls external bus buffers.
UDS, LDS	Data strobes	Identifies the byte(s) to be operated on according to R/W and AS.
DTACK	Data transfer acknowledge	Allows the bus cycle to synchronize with slow devices or memories.
BR	Bus request	Input to the processor from a device requesting the bus.
BG	Bus grant	Output from the processor granting bus arbitration.
BGACK	Bus grant acknowledge	Confirmation signal from BG indicating a valid selection from the arbitration process.
IPL0, IPL1, IPL2	Interrupt priority level	Provides the priority level of the interrupting function to the processor.
FC0, FC1, FC2	Function code	Provides external devices with information about the current bus cycle and processor state.
CLK	Clock	Master TTL input clock to the processor.
RES	Reset	Provides reset (initialization) signal to the processor and peripheral devices.
HLT	Halt	Stops the processor and allows single stepping.
BERR	Bus error	Provides termination of a bus cycle if no response or an invalid response is received.
E	Enable	Enable clock for 6800 systems.
VPA	Valid peripheral address	Identifies addressed area as a 6800 compatible area.
VMA	Valid memory address	Indicates to 6800 family devices that a valid address is on the bus.
V_{CC}	+5 volt (2 pins)	—
GND	Ground (2 pins)	—

FIGURE 2.20 Motorola 68000 pin diagram. (Source: Motorola. Courtesy Motorola Semiconductor Products, Incorporated.)

The program counter is separate, as is the status register shown in Figure 2.22. The various status fields are shown in Figure 2.22 as well.

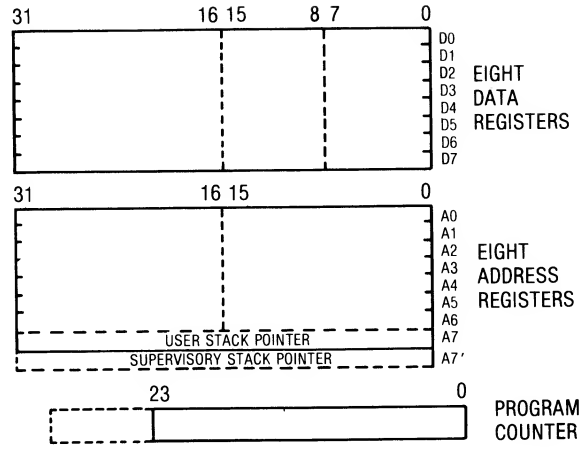


FIGURE 2.21 Motorola 68000 general-purpose registers. (Source: Motorola. Courtesy Motorola Semiconductor Products, Incorporated.)

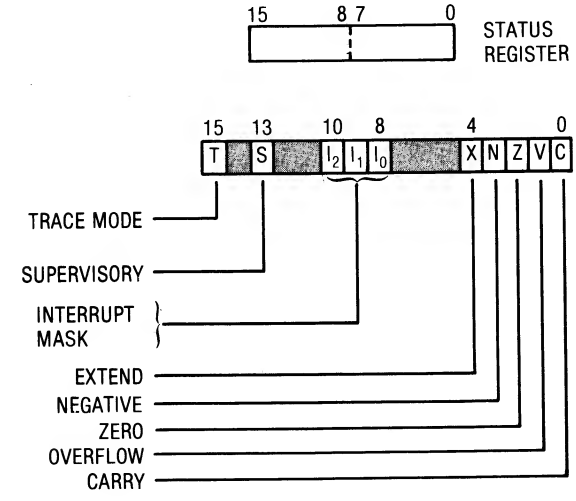


FIGURE 2.22 Motorola 68000 status register. (Source: Motorola. Courtesy Motorola Semiconductor Products, Incorporated.)

There are five basic types of addressing modes in the 68000:

- Register direct.
- Register indirect.
- Program counter relative.
- Absolute data.
- Immediate data.
- Implied.

These various addressing modes are summarized in

TABLE 2.4. MC68000—Data Addressing Modes

Mode	*Generation*
Register direct addressing	
Data register direct	EA = Dn
Address register direct	EA = An
Absolute data addressing	
Absolute short	EA = (next word)
Absolute long	EA = (next two words)
Program counter relative addressing	
Relative with offset	EA = (PC) + d_{16}
Relative with index and offset	EA = (PC) + (Xn) + d_8
Register indirect addressing	
Register indirect	EA = (An)
Postincrement register indirect	EA = (An), An← An + N
Predecrement register indirect	An← An - N, EA = (An)
Register indirect with offset	EA = (An) + d_{16}
Indexed register indirect with offset	EA = (An) + (Xn) + d_8
Immediate data addressing	
Immediate	DATA = next words(s)
Quick immediate	Inherent data
Implied addressing	
Implied register	EA = SR, USP, SP, PC

Notes:
EA = effective address
An = address register
Dn = data register
Xn = address or data register used as index register
SR = status register
PC = program counter
( ) = contents of
d_8 = eight-bit offset (displacement)
d_{16} = sixteen-bit offset (displacement)
N = 1 for byte, 2 for words and 4 for long words
← = replaces

SOURCE: Motorola Semiconductor Products Inc. Reprinted with permission.

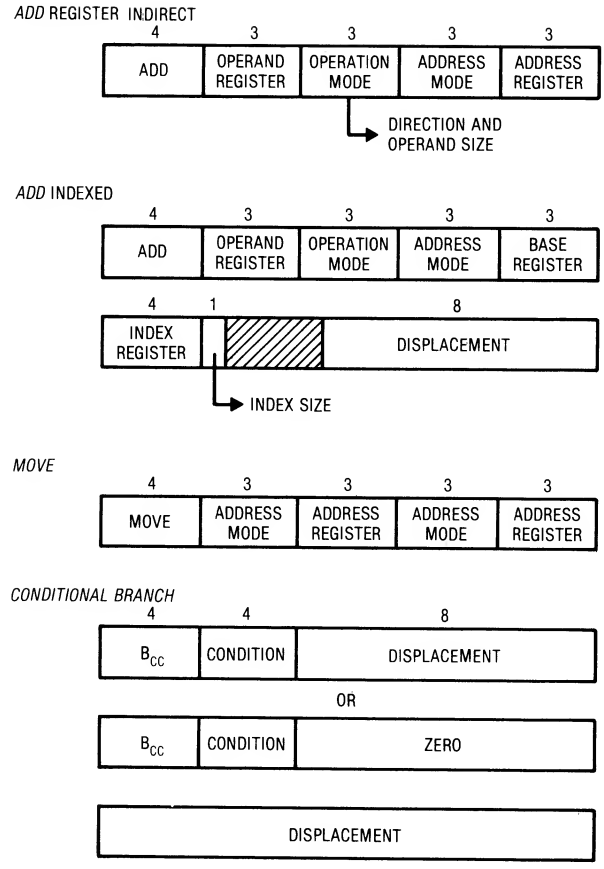


FIGURE 2.23 Motorola 68000 specific instruction formats. (Source: Motorola. Courtesy Motorola Semiconductor Products, Incorporated.)

Table 2.4. In addition, there is also bit addressing through the use of bit manipulation instructions (SET, CLEAR, CHANGE, and TEST). A memory word is first addressed using one of the addressing modes in Table 2.5, and then the individual bit is addressed by the bit number in that word.

The specification of the addressing mode is done by certain fields of the instruction. As an example, Figure 2.23 presents the format of four important instructions: ADD (register indirect mode), ADD (indexed mode), MOVE, and CONDITIONAL BRANCH.

The 68000 instructions are from one to five words in length and have the format shown in Figure 2.24. The first word of the instruction is called the operation word.

Most instructions specify the location of the operand by the effective address designations located in the operation word. Figure 2.25 shows the designation of the effective address by means of a mode and register fields in the operation word. The value in the mode field specifies the addressing mode and the value in the register field specifies the relevant register. Table 2.5 summarizes how the addressing mode is specified from the mode and register fields.

The address itself is a 24-bit quantity in either memory of an address register. This type of addressing implementation must be distinguished from the segment: offset addressing implementation found in the 8086 and the Z8000. No arbitrary segment sizes are required, and addressing is more straightforward. On the other hand, the advantages of segmentation for running different jobs or routines in different segments of memory, merely by changing the segment number, is an advantage of the 8086 and Z8000 architecture that is not found in the 68000.

Another advantage of the 68000 is that the address specification field in the microprocessor instruction is

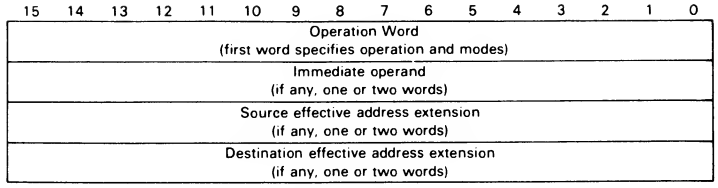

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Operation Word
(first word specifies operation and modes)
Immediate operand
(if any, one or two words)
Source effective address extension
(if any, one or two words)
Destination effective address extension
(if any, one or two words)

FIGURE 2.24 Motorola 68000 general instruction format. (Source: Motorola. Courtesy Motorola Semiconductor Products, Incorporated.)

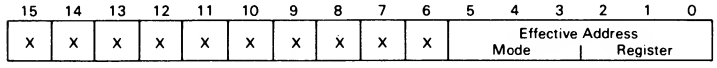


FIGURE 2.25 Motorola 68000 instruction operation word format. (Source: Motorola. Courtesy Motorola Semiconductor Products, Incorporated.)

TABLE 2.5. Effective Address Encoding Summary

Addressing Mode	*Mode*	*Register*
Data register direct	000	Register number
Address register direct	001	Register number
Address register indirect	010	Register number
Address register indirect with postincrement	011	Register number
Address register indirect with predecrement	100	Register number
Address register indirect with displacement	101	Register number
Address register indirect with index	110	Register number
Absolute short	111	000
Absolute long	111	001
Program counter with displacement	111	010
Program counter with index	111	011
Immediate or status register	111	100

only 6 bits, even though a larger address (24 bits in length) is actually being specified.

The instruction set of the 68000 is given in Table 2.6(a) and 2.6(b).

The 68000 permits operation in two states or modes: user and supervisory. The instruction set is divided into two types of instructions: common instructions to both the user and the supervisory states, and those instructions that can be operated only in the supervisory state. The distinction between states enables more sophisticated operating systems to be developed, and protects supervisory programs and memory from access by users. The ability to detect an unauthorized instruction (i.e., a supervisory instruction when the processor is in the user state) is known as a "trap," which is similar to an interrupt. The Z8000 also incorporates a user/supervisory mode and trap conditions, and further details on such operations are presented in connection with the Z8000 in Chapter 5. In the 68000, the trap instruction vectors, which find the appropriate entry point for the trap routine, are present in a low portion of memory adjacent to the interrupt vectors for entry to the various interrupt handlers, as shown in Figure 2.26.

Another important feature of the 68000 instruction set is a number of instructions specifically designed for high level language development. Some of these high-level language operations that have been simplified by the use of specific assembly language instructions are given below.

- Array accessing.
- Boolean expression evaluation.
- Establishment of new local environment.
- Limited precision arithmetic.
- Looping.
- Procedure calls and return.

These operations can be performed with specific instructions in Table 2.6.

- *Array accessing.* Array accessing is implemented with the use of CHK instruction (check register against bounds). A previously calculated array index is stored in a data register and checked against zero and a limit value addressed by the instruction. If the index is out of bounds for a particular array, a trap occurs.
- *Boolean expression evaluation.* The SCC (set conditionally) instruction functions to assign a true or false value to a Boolean variable based on the conditions used by the BCC (branch conditionally) in-

TABLE 2.6a. MC68000 Instruction Set[a]

Mnemonic	Description	Mnemonic	Description	Mnemonic	Description
ABCD	Add Decimal with Extend	EOR	Exclusive Or	PEA	Push Effective Address
ADD	Add	EXG	Exchange Registers	RESET	Reset External Devices
AND	Logical And	EXT	Sign Extend	ROL	Rotate Left without Extend
ASL	Arithmetic Shift Left	JMP	Jump	ROR	Rotate Right without Extend
ASR	Arithmetic Shift Right	JSR	Jump to Subroutine	ROXL	Rotate Left with Extend
Bcc	Branch Conditionally	LEA	Load Effective Address	ROXR	Rotate Right with Extend
BCHG	Bit Test and Change	LINK	Link Stack	RTE	Return from Exception
BCLR	Bit Test and Clear	LSL	Logical Shift Left	RTR	Return and Restore
BRA	Branch Always	LSR	Logical Shift Right	RTS	Return from Subroutine
BSET	Bit Test and Set	MOVE	Move	SBCD	Subtract Decimal with Extend
BSR	Branch to Subroutine	MOVEM	Move Multiple Registers	Scc	Set Conditional
BTST	Bit Test	MOVEP	Move Peripheral Data	STOP	Stop
CHK	Check Register Against Bounds	MULS	Signed Multiply	SUB	Subtract
CLR	Clear Operand	MULU	Unsigned Multiply	SWAP	Swap Data Register Halves
CMP	Compare	NBCD	Negate Decimal with Extend	TAS	Test and Set Operand
DBcc	Test Cond., Decrement and Branch	NEG	Negate	TRAP	Trap
DIVS	Signed Divide	NOP	No Operation	TRAPV	Trap on Overflow
DIVU	Unsigned Divide	NOT	One's Complement	TST	Test
		OR	Logical Or	UNLK	Unlink

TABLE 2.6b. MC68000 Variations of Instruction Types[a]

Instruction Type	Variation	Description	Instruction Type	Variation	Description
ADD	**ADD**	Add	**MOVE**	**MOVE**	Move
	ADDA	Add Address		MOVEA	Move Address
	ADDQ	Add Quick		MOVEQ	Move Quick
	ADDI	Add Immediate		MOVE from SR	Move from Status Register
	ADDX	Add with Extend		MOVE to SR	Move to Status Register
				MOVE to CCR	Move to Condition Codes
AND	**AND**	Logical And		MOVE USP	Move User Stack Pointer
	ANDI	And Immediate	**NEG**	**NEG**	Negate
				NEGX	Negate with Extend
CMP	**CMP**	Compare	**OR**	**OR**	Logical Or
	CMPA	Compare Address		ORI	Or Immediate
	CMPM	Compare Memory	**SUB**	**SUB**	Subtract
	CMPI	Compare Immediate		SUBA	Subtract Address
				SUBI	Subtract Immediate
EOR	**EOR**	Exclusive Or		SUBQ	Subtract Quick
	EORI	Exclusive Or Immediate		SUBX	Subtract with Extend

[a] *Source: Motorola Semiconductor Products, Inc. Reprinted by permission.*

struction. This procedure is useful in implementing Boolean expression evaluation.

- *Establish new local environment.* The establishment of a new local environment is important during procedure calls. The LINK (link stack) instruction saves the old contents of the frame pointer on the stack, points the new frame pointer to the top of the stack, and subtracts the appropriate number of bytes from the stack pointer as required by the procedure. Thus local storage, or a new local environment, is easily established for the called procedure.
- *Limited precision arithmetic.* The TRAPV (trap on overflow) instruction permits overflow testing by causing a trap if the preceding operation caused an overflow.
- *Looping.* The DBCC (decrement and branch conditionally) instruction is highly useful in constructing simple loops.
- *Procedure calls and return.* The procedure call using 68000 instructions is shown in Figure 2.27, together with the equivalent program in Pascal. Figure 2.28 shows the stack before and after the procedure call, with the location of the frame pointer (FP) and the stack pointer (SP).

It would be worthwhile to describe the operation of the simple 68000 program shown in Figure 2.27. First, the MOVE instruction pushes a value onto the stack. The PEA (push effective address) instruction then pushes the result of an arbitrary address calculation onto

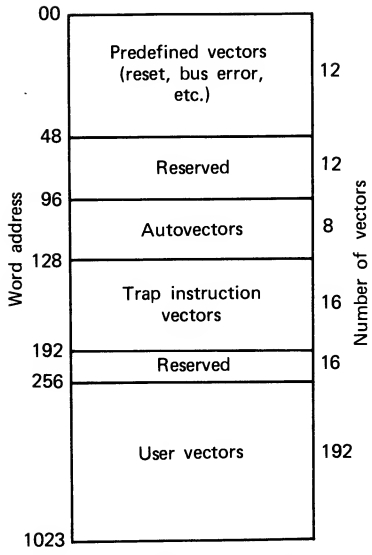


FIGURE 2.26 Motorola 68000 trap and interrupt vector memory map. (Source: Motorola. Courtesy Motorola Semiconductor Products, Incorporated.)

the stack. This instruction permits the programmer to precalculate addresses and later call them by reference. The JSR (jump to subroutine) instruction then pushes the return address onto the stack and jumps to the procedure entry point. The LINK instruction has been described above. the MOVEM (move multiple registers)

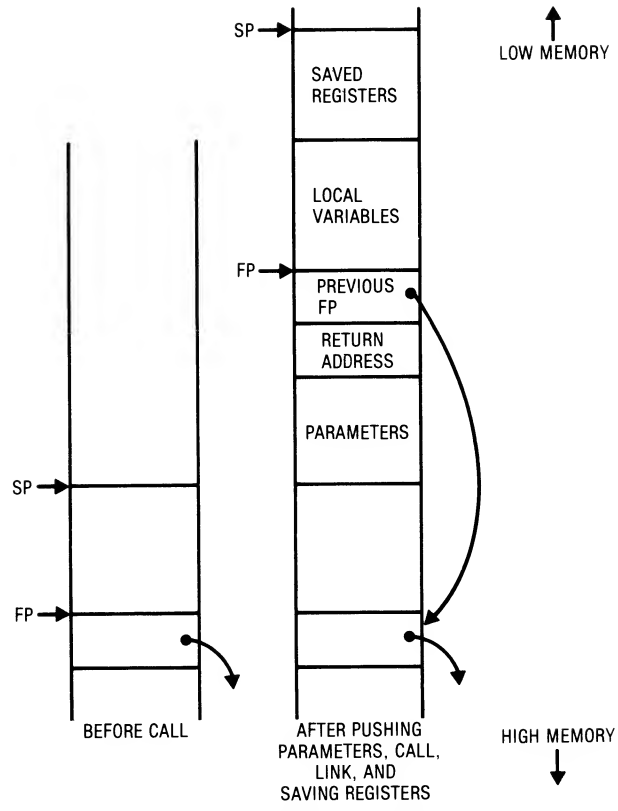


FIGURE 2.28 Motorola 68000 procedure call. (Source: Motorola. Courtesy Motorola Semiconductor Products, Incorporated.)

SAMPLE PROGRAM

```
PROGRAM EXAMPLE;
VAR PARAM1, PARAM2: INTEGER;
PROCEDURE PROC (X: INTEGER; VAR Y: INTEGER);
  VAR A, B: INTEGER;
  BEGIN
  <procedure body>
  END;
BEGIN
  PROC (PARAM1, PARAM2)
END
```

PROGRAM BODY

```
MOVE      PARAM1 TO -(SP)      "push first parameter"
PEA       PARAM2               "push address of second parameter"
JSR       PROC                 "call the procedure"
ADD       #6 TO SP             "pop parameters from the stack"
```

PROCEDURE BODY

```
LINK      FP, 4                "link and allocate three local variables"
MOVEM     <registerlist> TO -(SP)   "push some register contents"
<procedure body>
MOVEM     <registerlist> FROM (SP) +   "restore registers"
UNLK      FP                   "restore stack"
RETURN                         "return to calling procedure"
```

SOURCE: Motorola. Reprinted with permission.

Figure 2.27. Pascal program and equivalent M 68000 program.

instruction saves an arbitrary subset of the registers on the stack, the particular registers being designated by setting corresponding bits in a 16-bit field of the instruction. Finally, the UNLK (unlink stack) instruction undoes the LINK instruction, and the RETURN instruction pops the return address from the stack and returns the program to the calling procedure.

Ferranti F100-L

The Ferranti F100-L is a 16-bit, single-chip microprocessor implemented with the Ferranti bipolar collector diffusion isolation (CDI) process. Its development was sponsored by the U.K. Ministry of Defense, and it was first marketed in 1977 as an OEM product.

The most significant feature of the F100-L is its technology. In using the nonsaturation CDI gate, it is possible to achieve propagation delays as low as 3 nsec, and speed-power products of less than 1 pJ. The chip employs a single phase clock at a frequency of 8 MHz, and typical instruction execution is 3–4 μsec. The chip is 0.23 in. (5.8 mm) square, dissipates less than 375 mwatts, and operates over the full military specification operating temperature range of −55 to +125°C. Control and data interface chips, as well as analog/digital and digital/analog converters, are also under development with the same CDI process. All the devices use a single +5 V supply and are TTL compatible.

The architecture of the F100-L includes serial function and register control units; parallel register control; operand, accumulator, instruction, and condition registers; a program counter; executive decode and control; and I/O transmitter/receiver circuitry.

The addressing modes are: direct addressing (of the first 2K words); pointer indirect (with optional auto decrement or increment), immediate indirect, and immediate data. Double length instructions are also included.

The F100-L contains an instruction set that includes external function enable instructions, so that external special-purpose hardware units are activated. These include fast multiply and divide, trigonometric functions, or more complex operations such as Fourier transforms.

Special-purpose interface chips, the F111-L and F112-L, are also provided to interface the F100-L to memory and peripherals. Facilities include daisy chain priority for multiple DMA, and full vector interrupt capabilities.

Data General mN 602

The microNOVA mN 602 is a 16-bit NMOS microprocessor packaged in a 40-pin package. The basic block diagram is shown in Figure 2.29, and the pin diagram in Figure 2.30. The registers labeled in the figure are as follows:

ACO	Accumulator 0	16 bits
AC1	Accumulator 1	16 bits
AC2	Accumulator 2	16 bits
AC3	Accumulator 3	16 bits
SP	Stack pointer	15 bits
FP	Frame pointer	15 bits
PC	Program counter	15 bits
C	Carry	1 bit
ION	Interrupt enable	1 bit
RTC ON	Real-time clock enable	1 bit
RTC RQ	Real-time clock request	1 bit
SO RQ	Stack overflow request	1 bit
RAC	Refresh address counter	7 bits

The accumulators are general-purpose data registers; AC2 and AC3 may also be used as index registers. The stack pointer has its usual meaning; the frame pointer points to an area in the user stack containing the current procedure's local variables; the program counter, carry bit, interrupt enable, and stack overflow have their usual meanings. The real-time clock request indicates that the real-time clock is requesting an interrupt. The refresh address counter is a 7-bit register that contains the address of the next row of memory instructions to be refreshed.

An expanded block diagram of the mN 602 arithmetic unit is shown in Figure 2.31.

There are six classes of instructions in the instruction set of the mN 602:

- Memory reference class.
- Arithmetic and logic class.
- Input/output class.
- Multiply/divide class.
- Stack manipulation class.
- Central processor control class.

A listing of the instruction set is provided in Table 2.7.

The calculation of the effective address is performed according to the flowchart shown in Figure 2.32.

Data General has implemented the mN 602 both as a single board computer—the MBC/1 shown in Figure 2.33—and as a microcomputer system known as the MP/100 family, shown in Figure 2.34.

The MBC/1 combines the microprocessor with 2K-bytes static RAM, sockets for 4K-bytes PROM, an asynchronous interface, and a 32-line digital I/O port.

The MP/100 includes further features such as a real-

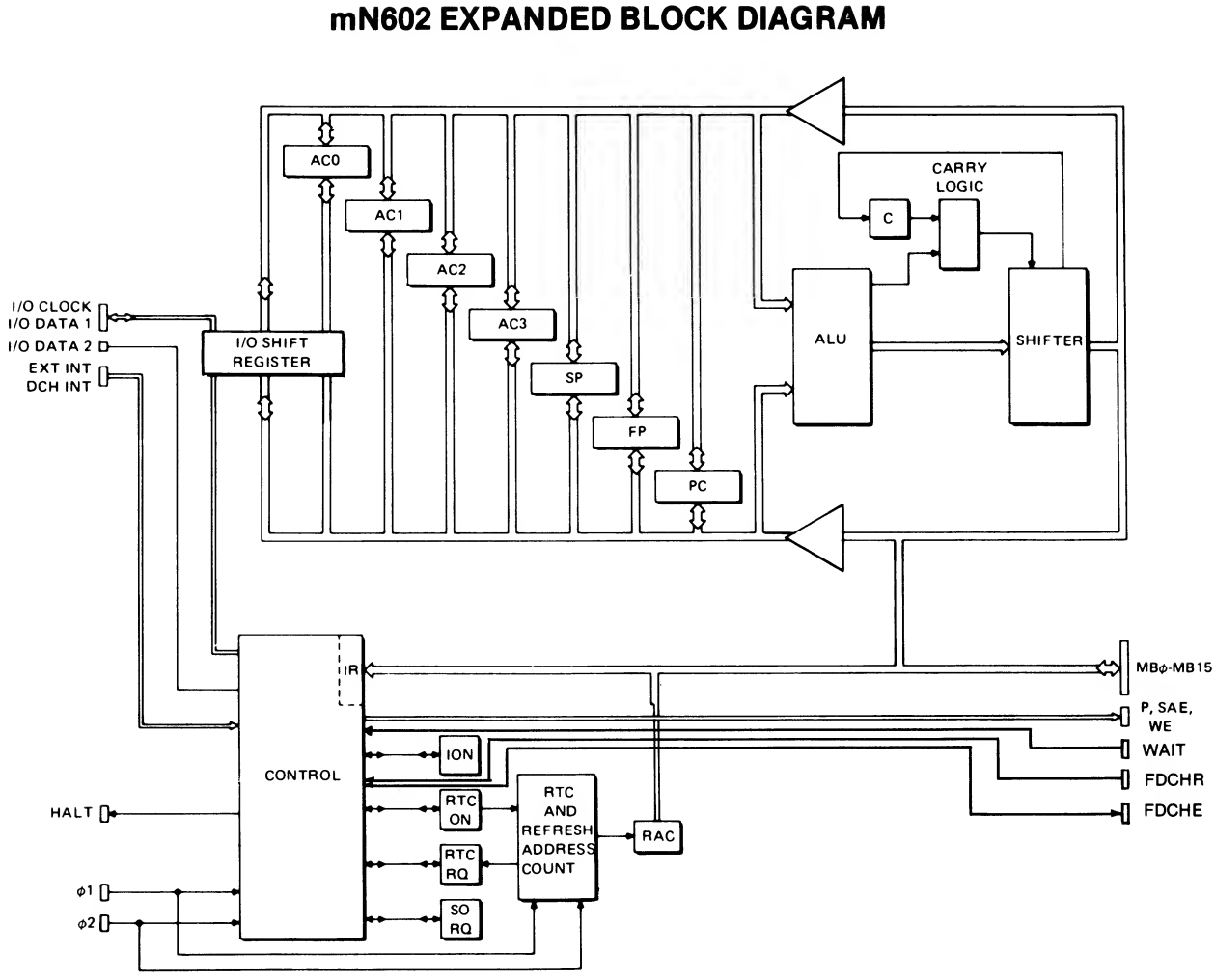


FIGURE 2.29 Data General mN 602 block diagram. (Source: Data General. microNOVA is a Trademark of Data General Corporation, Westboro, Mass.)

time clock, a power fail/auto restart circuit, and automatic program load.

For those applications requiring faster execution than can be provided with MOS technology, Data General also provides the MP/200 computer family based on bipolar circuitry for use rather than the mN 602 microprocessor. As a point of comparison, the add time in the mN 602 is 2.4 μsec, and 840 nsecs in the MP/200 system.

National 16000 Family

The National 16000 family is the latest entry of the major semiconductor manufacturers in the 16-bit market and is expected to be available in sample quantities in late 1980. The NS16000 family presently includes three members—the 16008, 16016, and 16032. The devices are expected to be fabricated in X-MOS, National's scaled MOS process, and packaged in a 48-pin package.

The NS16008 is designed with 16-bit internal architecture, but with an external 8-bit bus. The 16016 has the same architecture, but a 16-bit external bus. Finally, the 16032 has an internal 32-bit architecture (i.e., 32-bit ALU and registers) and a 16-bit external bus.

One of the important features of the devices is expected to be compatibility with the 8080 instruction set, which will be emulated in software.

Advanced Micro Devices 29116

The AMD 29116 is a 16-bit ECL bipolar microprocessor announced in late 1979 and expected to become available in late 1980. It is packaged in a 48-pin package.

The 29116 has a 16-bit ALU, a 16-bit accumulator,

PIN NUMBERS AND NAMES

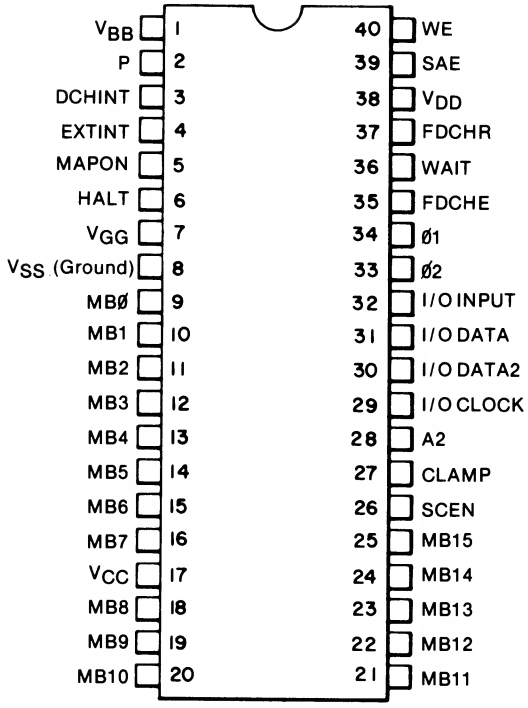


FIGURE 2.30 Data General mN 602 pin diagram. (Source: Data General. microNOVA is a trademark of Data General Corporation, Westboro, Mass.)

ORGANIZATION OF ARITHMETIC UNIT

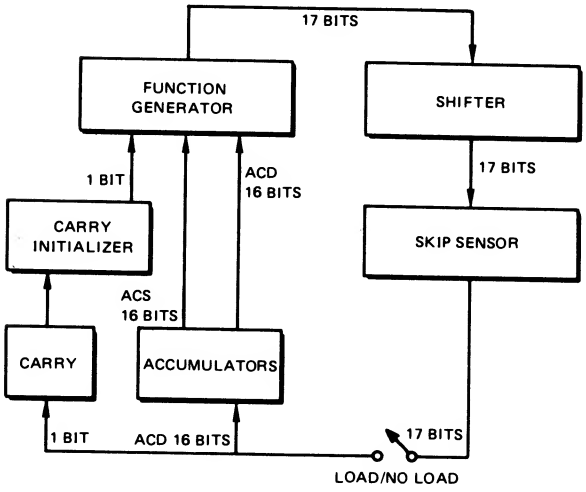


FIGURE 2.31 Data General mN 602 arithmetic unit block diagram. (Source: Data General. microNOVA is a trademark of Data General Corporation, Westboro, Mass.)

TABLE 2.7. mN 602 Instruction Set

Memory Reference Instructions (MRI)

No Accumulator

OP CODE

O	O	O	OP	@	INDEX	DISPLACEMENT
0	1	2	3 4	5	6 7	8 9 10 11 12 13 14 15

In the No Accumulator–MRI format instructions, bits 0-2 are 000, and bits 3-4 contain the operation code. The effective address "E" is computed from bits 5-15 as shown in Figure 2.32.

	OP	
JMP	00	Jump to location E (put E in PC)
JSR	01	Load PC + 1 in AC3 and jump to subroutine at location E (put E in PC)
ISZ	10	Increment location E by 1 and skip if result is zero
DSZ	11	Decrement location E by 1 and skip if result is zero

One Accumulator

OP CODE

O	OP	AC	@	INDEX	DISPLACEMENT
0	1 2	3 4	5	6 7 8	9 10 11 12 13 14 15

In the One Accumulator–MRI format instructions, bit 0 is 0, and bits 1-2 contain the operation code. Bits 3-4 specify the accumulator for the operation. The effective address is computed from bits 5-15 as shown in Figure 2.32

	OP	
LDA	01	Load contents of location E into AC.
STA	10	Store AC in location E.

Arithmetic and Logical Instructions (ALC)

1	ACS	ACD	OP	SH	C	#	SKIP
0	1 2	3 4	5 6 7	8 9	10 11	12	13 14 15

In the ALC format instructions, bit 0 is 1, bits 1 and 2 specify the source accumulator, bits 3-4 specify the destination accumulator, bits 5-7 contain the operation code, bits 8 and 9 specify the action of the shifter, bits 10 and 11 specify the value to which the carry bit will be initialized, bit 12 specifies whether or not the result will be loaded into the destination accumulator, and bits 13-15 specify the skip test. Each instruction in this format utilizes an arithmetic unit whose logical organization is shown in Figure 2.31. Each instruction specifies two accumulators to supply operands to the function generator, which performs the function specified by bits 5-7 of the instruction. The function generator also produces a carry bit whose value depends upon three quantities: an initial value specified by the instruction, the inputs, and the function performed. The initial value can be derived from the previous value of the carry bit, or the instruction can specify an independent value.

The 17-bit output of the function generator, made up of the carry bit and the 16-bit function result, then goes to the shifter, where it can be rotated one place right or left, or the two 8-bit

(continued)

TABLE 2.7. (*Continued*)

halves of the function result can be swapped without affecting the carry bit. The 17-bit output of the shifter can then be tested for a skip. The skip sensor can test whether the carry bit or the rest of the 17-bit result is or is not equal to zero. After being tested by the skip sensor, the shifter output can be loaded into the carry bit and the destination accumulator. An instruction in this format can perform a complicated arithmetic and shifting operation, and test the result for a skip without affecting the carry bit or either of the operands.

	OP	
COM	000	Place the complement of ACS in ACD
NEG	001	Place negative of ACS in ACD
MOV	010	Move ACS to ACD
INC	011	Place ACS + 1 in ACD
ADC	100	Add the complement of ACS to ACD
SUB	101	Subtract ACS from ACD
ADD	110	Add ACS to ACD
AND	111	And ACS with ACD

Input/Output Instructions (I/O)

CONTROL

0	1	1	AC	OP	CTRL	DEVICE CODE

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

In the Input/Output format instructions, bit 0-2 are 011, bits 3-4 specify the accumulator for the operation, bits 5-7 contain the operation code, bits 8-9 select a control pulse or a test condition and bits 10-15 contain the device code of the referenced device.

	OP	
NIO	000	No operation.
DOA	010	Data out, AC to A buffer
DIB	011	Data in B buffer to AC
DOB	100	Data out, AC to C buffer
DIC	101	Data in, C buffer to AC
DOC	110	Data out, AC to C buffer
SKP	111	Skip

Multiply/Divide Instructions (M/D)

OP CODE

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	1	1	0	1	1	0	OP		0	0	0	0	0	1

	OP	
DIV	01	If AC0-AC2, set carry and abort the operation. Otherwise, clear carry and divide the 32-bit integer located in AC0 (most significant) and AC1 (least significant) by AC2. The 16-bit quotient is placed in AC1 and the 16-bit remainder is placed in AC0
MUL	11	Multiply the 16-bit integers located in AC1 and AC2 and ADD AC0 to then 32-bit product. The most significant 16 bits of this result are placed in AC0; the least significant in AC1

TABLE 2.7. (*Continued*)

Stack Manipulation Instruction (STK)

0	1	1	AC	OP	0	0	0	0	0	1

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

Stack support in the microNOVA processor consists of eight instructions.

	OP	
MTFP	00000	Move contents of AC to frame pointer.
MFFP	00010	Move contents of frame pointer to AC
MTSP	01000	Move contents of AC to stack pointer
MFSP	01010	Move contents of stack pointer to AC
PSHA	01100	Increment stack pointer and move contents of AC to top of stack
POPA	01110	Move top word on stack to AC and decrement stack pointer
SAV	10100	Push a 5-word return block on STACK

Central Processor Control Instructions (CPU)

0	1	1	0	0	OPCODE	1	1	1	1	1	1

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

I/O instructions with a device code of 77 perform a number of special functions rather than controlling a specific device. In all but the I/O SKIP instruction, I/O instructions with a device code of 77 use bits 8-9 to control the condition of the Interrupt On flag. An I/O SKIP instruction with a device code of 77 uses bits 8-9 to test the state of the Interrupt On Flag.

HALT	063077	Halt the processor
INTA	061477	Acknowledge Interrupt by loading code of the highest priority device requesting an interrupt into AC bits 10-15
INTDS	060277	Disable interrupt by clearing Interrupt On
INTEN	060177	Enable interrupt by setting Interrupt On
IORST	061077	Clear all I/O devices (=DOAC0, CPU)
MSKO	062077	Set up Interrupt Disable flags according to mask in AC
RTCEN	071077	Enable interrupts from CPU real-time clock
RTCDS	065077	Disable Interrupts from CPU real-time clock

SOURCE: Data General, Inc. Copyright 1979 Data General Inc. Reprinted by permission. microNOVA is a trademark of Data General Corp., Westboro, Ma.

a 32 word X 16 bit RAM, and a 16-bit data latch. The chip is specified to operate at a 100 nsec microcycle time. The instruction set includes a number of special purpose instructions, such as a cyclic redundancy check (CRC) instruction.

The 29116 has been designed to interface with many of the components of the AMD 2900 series, such as the

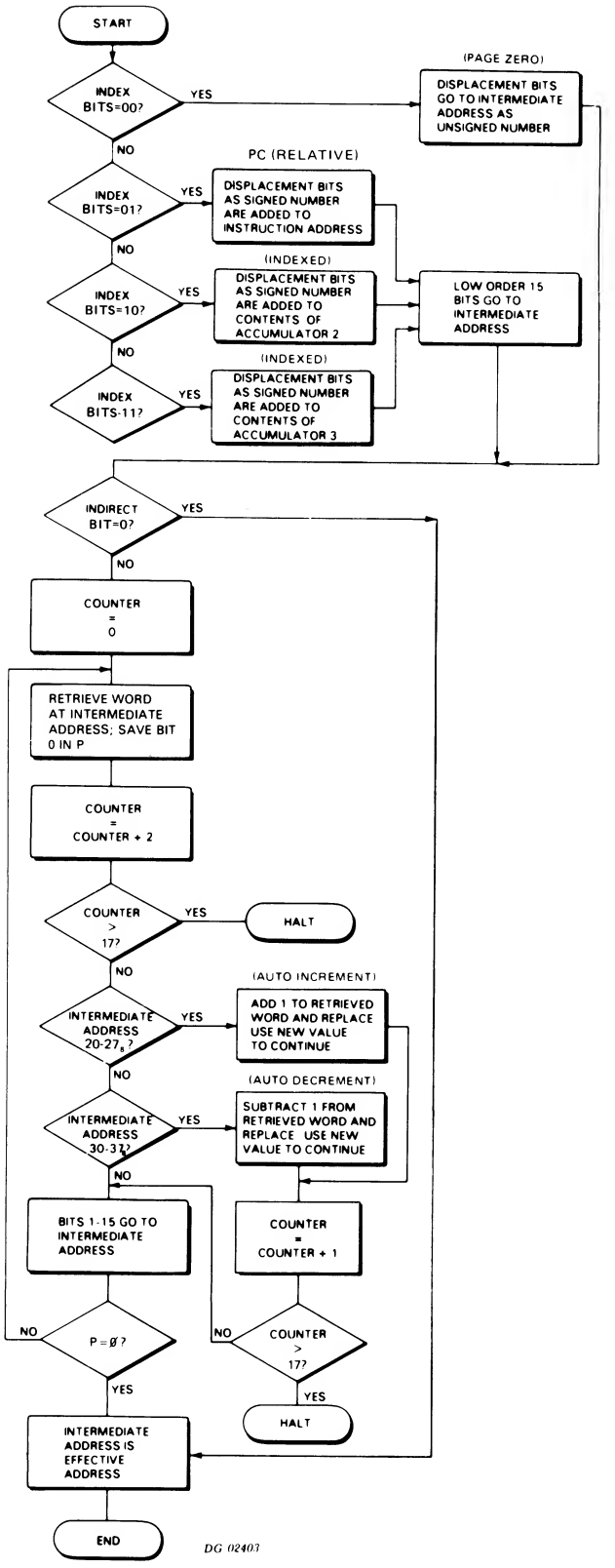


FIGURE 2.32 Data General mN 602 effective memory address calculation. (Source: Data General. microNOVA is a trademark of Data General Corporation, Westboro, Mass.)

2910 microprogram controller, which is described in greater detail in Chapter 7.

Eight-Bit Microprocessors with 16-Bit Internal Architecture

In addition to the fully implemented, true 16-bit microprocessors described above, it is important to make note of some recently announced 8-bit microprocessors that are designed with 16-bit internal architecture. These devices perform essentially the same operations as the 16-bit counterparts, but have an external 8-bit address and data bus. Examples of such devices include the Intel 8088 (counterpart to the Intel 8086) and the Motorola 6809 (counterpart to the Motorola 68000). Such products fit in the microprocessor product spectrum between the upper-end 8-bit microprocessors and the 16-bit microprocessors. They offer the user the expanded software capabilities of the 16-bit architecture while retaining the 8-bit interface structure.

Intel 8088

The Intel 8088 is an 8-bit microprocessor with a 16-bit internal architecture and is implemented using n-channel HMOS technology. The functional block diagram of the 8088 is shown in Figure 2.35, and the pin diagram of the 40-pin package in Figure 2.36. A photomicrograph of the Intel 8088 is presented in Figure 2.37.

The internal architecture of the 8088 is basically the same as the 8086, as can be seen by comparing Figure 2.35 with Figure 2.1. The basic distinction is that the 8088 has an 8-bit external data bus. There are also some minor internal operational differences related to the different bus widths—the queue length is 4 bytes in the 8088, compared to 6 bytes in the 8086, for example. Moreover, the instruction fetching is handled differently in the 8088, resulting in longer execution time in the 8088 for 16-bit operations due to the multiplexing overhead.

Since most of the internal functions of the 8088 are identical to the 8086, the detailed explanations of the internal registers and the instruction set in the 8086 presented in the next chapters are equally applicable to the 8088. The 8088 and the 8086 are software compatible provided such software is not system dependent.

Since the 8088 utilizes an external 8-bit data bus, the system configuration for the 8088 is different from that of the 8086. Although maximum and minimum mode 8086 systems are defined in the next chapter, it is worthwhile to illustrate the 8088 versions of such systems at this point for completeness. Figure 2.38 illustrates a typical minimum mode 8088 system, while Figure 2.39 shows a typical maximum mode 8088 system.

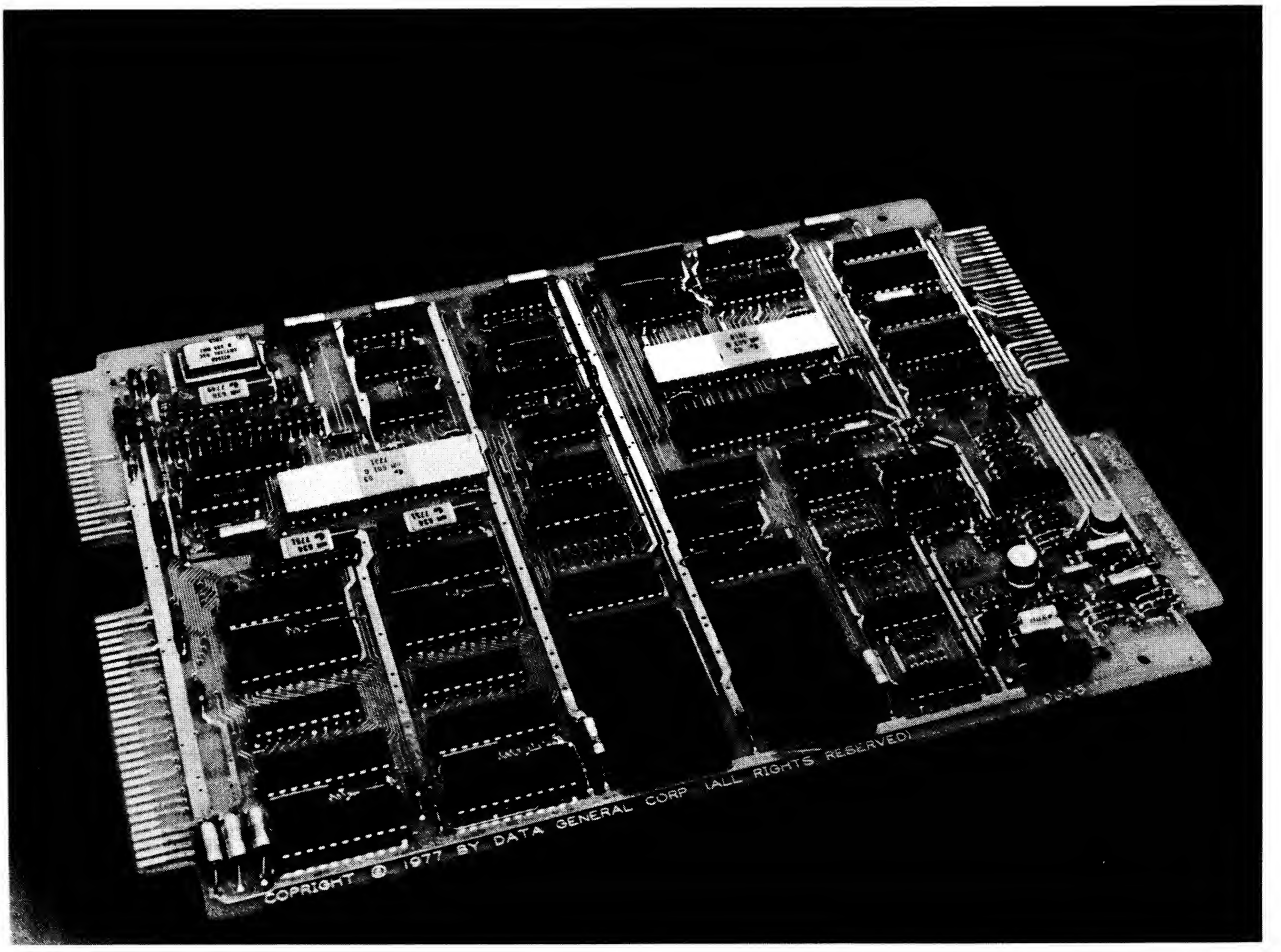


FIGURE 2.33 Data General MBC/1 single board computer. (Courtesy Data General. microNOVA is a trademark of Data General Corporation, Westboro, Mass.)

Future 32-bit Microprocessors

Although 16-bit microprocessors are presently still at the stage of being designed into systems, the next generator of 32-bit microprocessors is on the drawing boards of semiconductor manufacturers, and the basic architectural designs of such processors are being preannounced.

Intel Corporation has described a three-chip 32-bit processor, designated the iAPX 432, consisting of a data processor, a storage module, and an interface processor. Prior to the introduction of the iAPX 432, the company plans extensions of the 8086 and 8088, to be called the iAPX 186 and iAPX 188, respectively. Bridging the gap between the 8086/8088 extensions and the iAPX 432, the company plans a processor that will be called the iAPX 286. Although the time for introducing the last two devices has not been determined, the iAPX 186 and iAPX 188 extensions of the 8086 and 8088 are expected sometime in 1981.

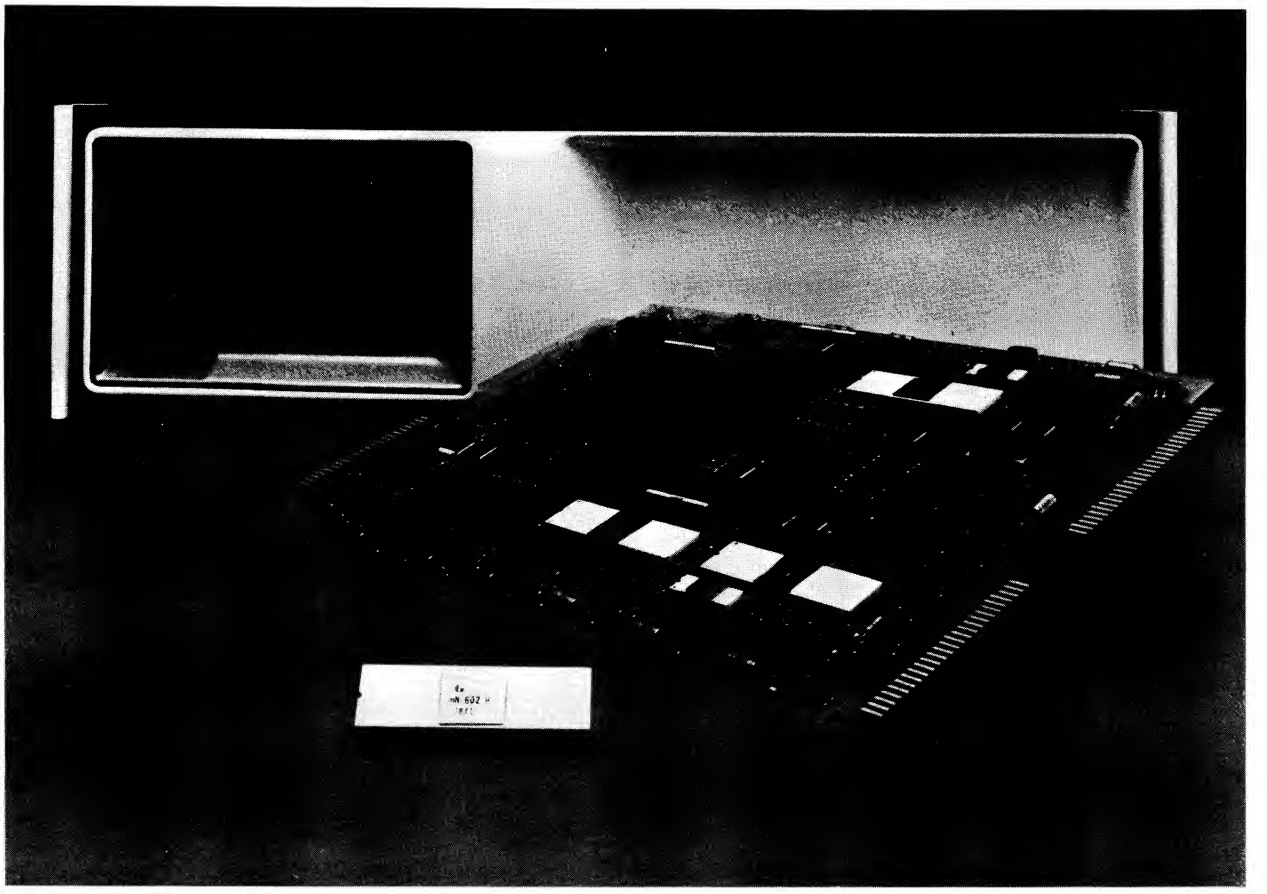


FIGURE 2.34 Data General MP/100 system. (Courtesy Data General. microNOVA is a trademark of Data General Corporation, Westboro, Mass.)

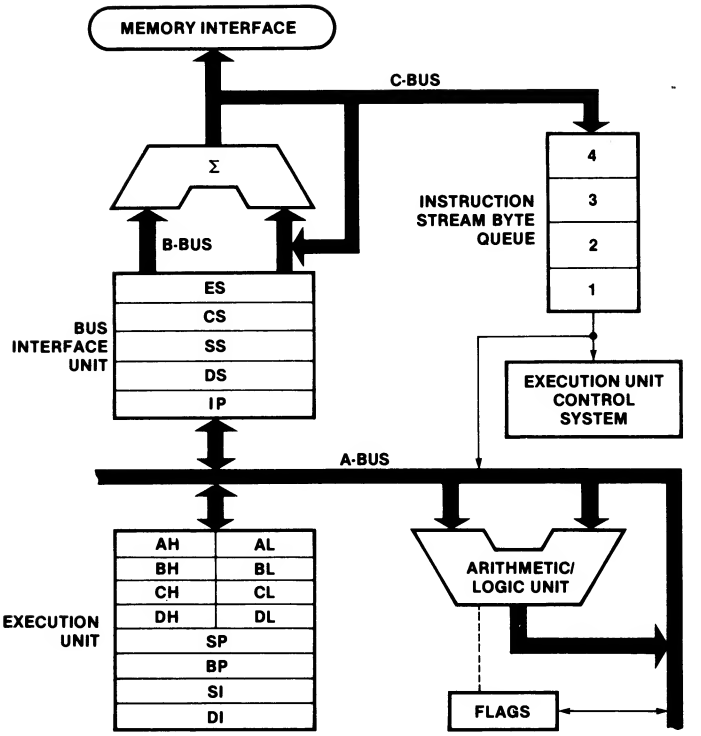


FIGURE 2.35 Intel 8088 functional block diagram. (Source: Intel Corporation.)

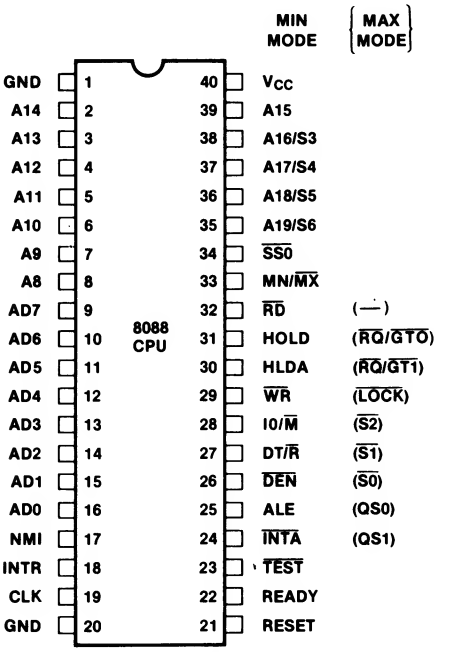


FIGURE 2.36 Intel 8088 pin diagram. (Source: Intel Corporation.)

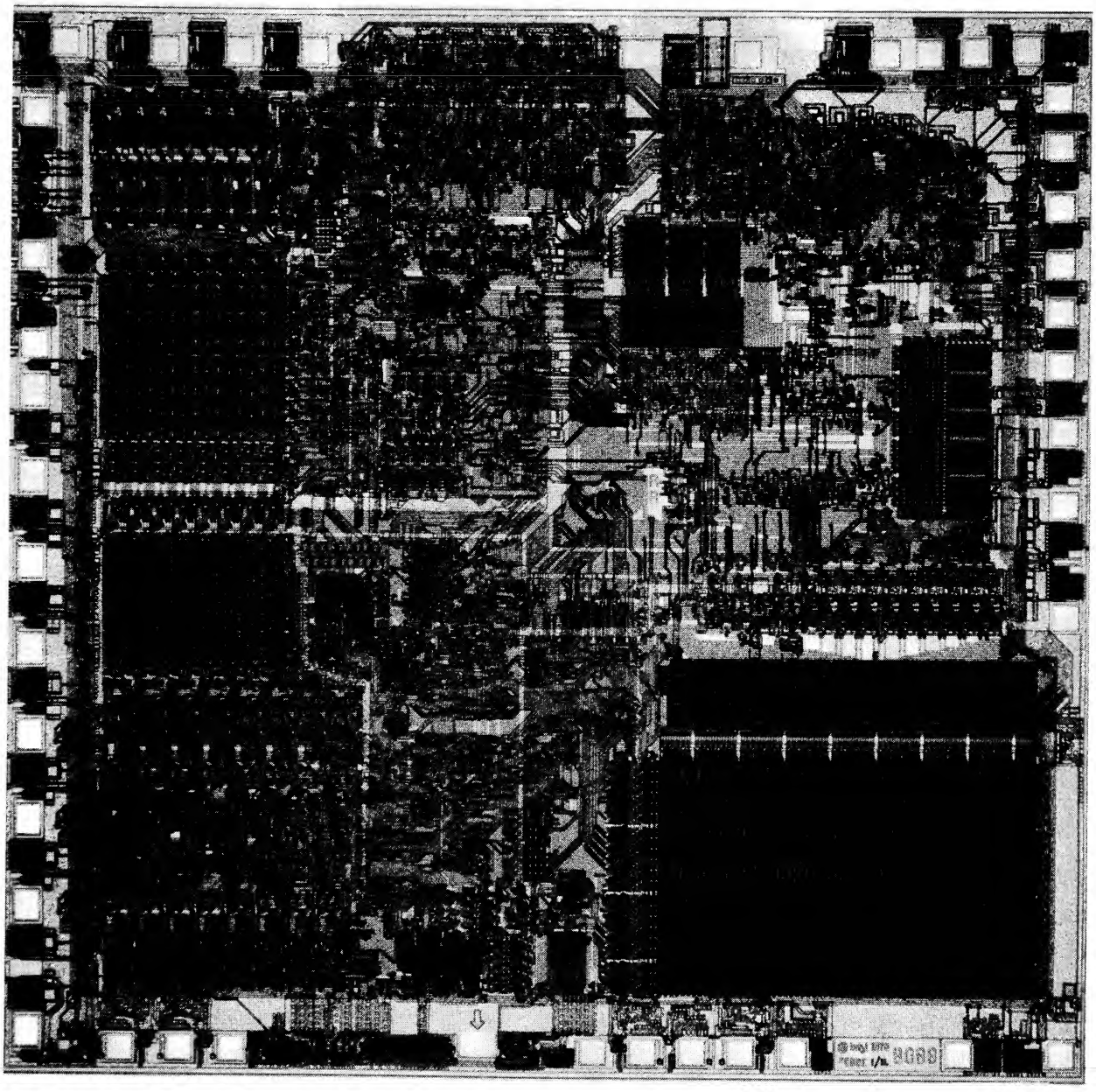


FIGURE 2.37. Intel 8088 (Reprinted with permission. Courtesy Intel Corporation.)

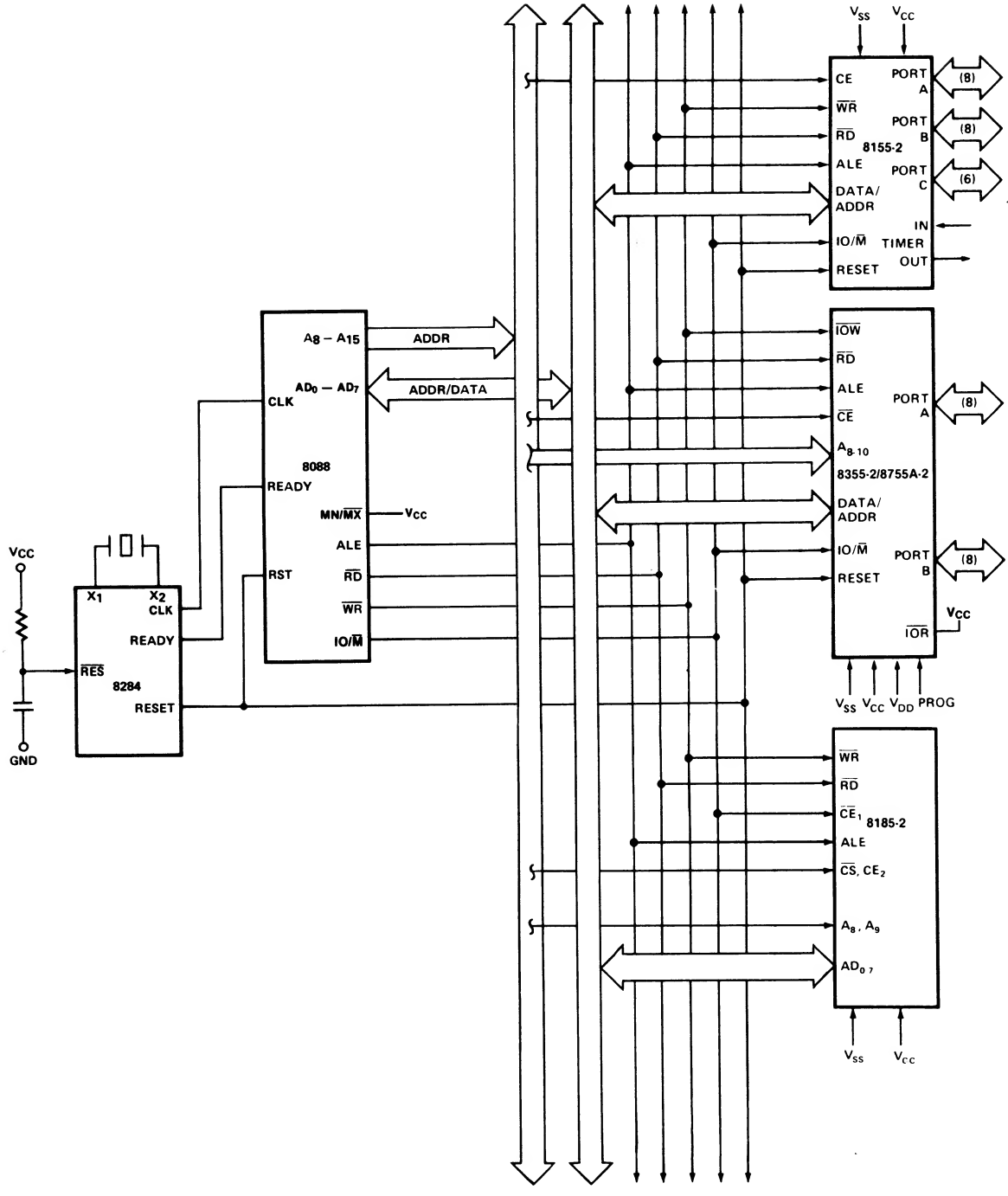


FIGURE 2.38 Intel 8088 multiplexed bus configuration, minimum mode. (Source: Intel Corporation.)

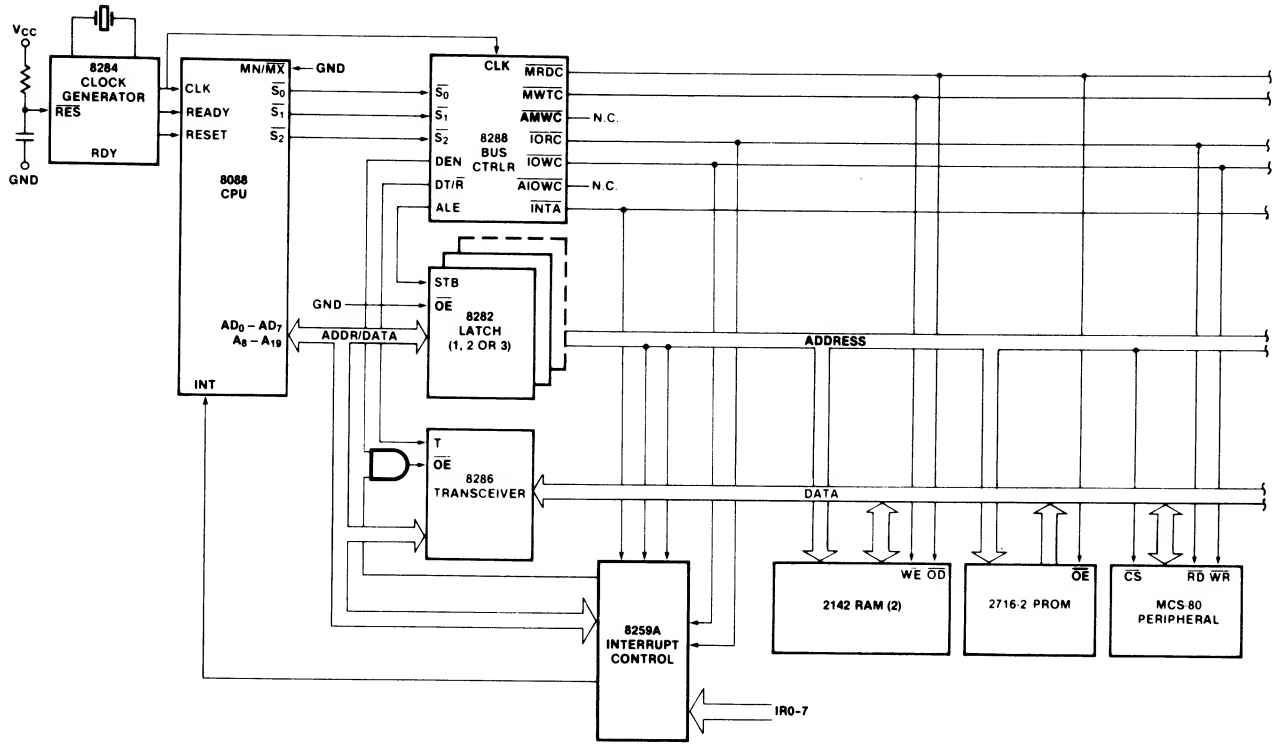


FIGURE 2.39 Intel 8088 maximum mode system. (Source: Intel Corporation.)

3

intel 8086 system

Since the Intel 8086 is one of the more important of the new 16-bit microprocessors, it would be worthwhile examining the operation and system design using that processor in greater detail. In this chapter, we consider the following features:

- Internal architecture.
- System configuration.
- System timing.
- Instruction set.

Internal Architecture

The internal architecture of the 8086 can be described in terms of the detailed functional block diagram shown in Figure 3.1. As we point out in Chapter 2, the processor is divided into two separate and distinct units: a bus interface unit (BIU) and an execution unit (EU).

The BIU contains a C bus, an instruction stream byte queue, and the Segment Register File (CS, DS, SS, ES). The BIU also contains the instruction pointer (IP). The EU contains the General Register File (AH, AL, BH, BL, CH, CL, DH, DL, SP, BP, SI, DI), the arithmetic logic unit, and the operand and flag file.

The general registers are shown in Figure 3.2. The two 8-bit registers AH and AL together form a 16-bit register called AX; the two 8-bit registers BH and BL together form a 16-bit register called BX; and CX and DX are defined similarly. These 16-bit registers may also be referred to as follows:

AX: accumulator.

BX: base.

CX: count.

DX: data.

The general registers may be addressed either as 16-bit registers or as 8-bit registers. The latter capability makes the 8086 able to handle 8080 instructions with merely a code translation. In other words, the 8080 registers are a subset of the 8086 registers, and programs written for the 8080 can be translated to the 8086 without that much difficulty. Specifically, the AL register in the 8086 corresponds to the AC register in the 8080. The ADD B instruction in the 8080 (add the B register to the accumulator) translates into ADD AL, BH (add the BH register to the accumulator) translates into ADD AL, BH (add the BH register to the AL register) in the 8086.

It is also interesting to compare the speed of execution of instructions on the 8080 and the 8086. The register-to-register add instruction on the 8080 takes 4 clock

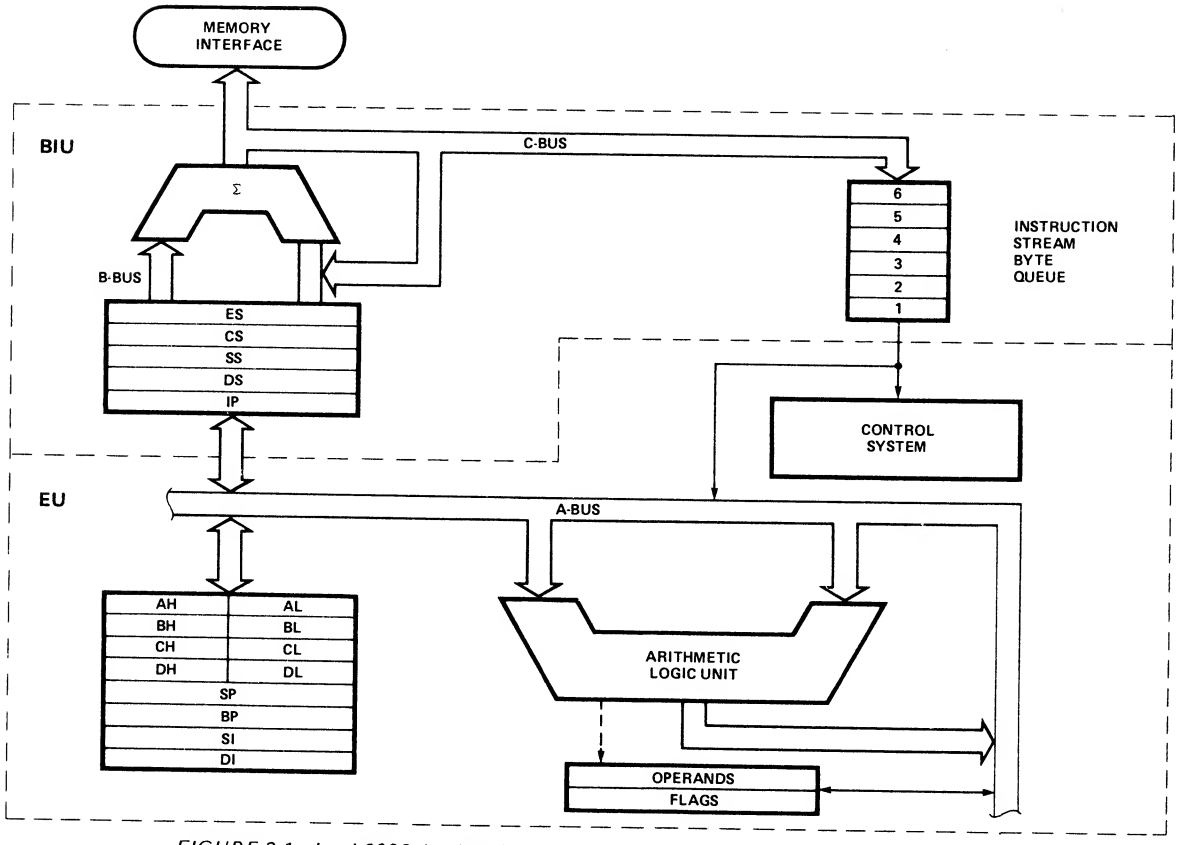


FIGURE 3.1 Intel 8086 detailed functional block diagram. (Source: Intel Corporation.)

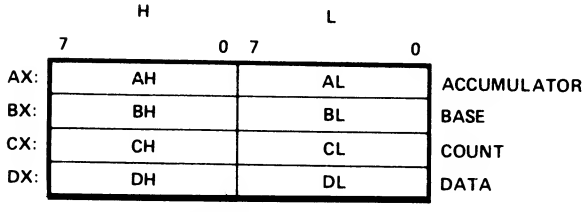


FIGURE 3.2 Intel 8086 general registers. (Source: Intel Corporation.)

cycles; depending on the clock rate, this could amount to between 1.3 and 2.0 μsecs. The register-to-register add instruction on the 8086 takes 3 clock cycles; with a 5-MHz clock, this amounts to 600 nsec, while with an 8-MHz clock, it would take 375 nsec. It must also be remembered that the 8080 is an 8-bit add, while the 8086 is a 16-bit add.

Also shown in Figure 3.2 is the pointer and index register file: SP, BP, SI, DI. These registers contain offset addresses for addressing within a segment. They also have certain default meanings, that is, meanings automatically assigned to them not specified by a program instruction. The pointers are assumed to contain offset addresses within a current stack segment, while the indexes are assumed to contain offset addresses within the current data segment.

The flag file, or flag register, is shown in Figure 3.3. The flags are all 1-bit registers and are used for control purposes. Their functions are generally indicated by the memonics used for them:

AF: auxiliary carry.
CF: carry.
DF: direction.
IF: interrupt enable.
OF: overflow.
PF: parity.
SF: sign.
TF: trap.
ZF: zero.

Some of the flag bit registers are identical to those used in the 8080. It could be said that the low-end 8 bits of the flag register in the 8086 is equivalent to the PSW register in the 8080, since the S, Z, -, A, -, P, -, and C flags are contained therein (wherein "-" denotes an undefined flag bit).

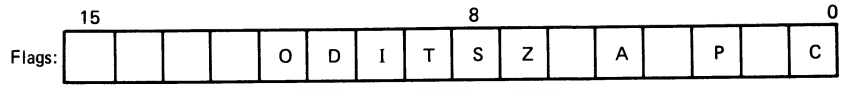


FIGURE 3.3 Intel 8086 flag registers.

As we see later in an analysis of the instruction set of the 8086, certain instructions treat the flag register as a 16-bit register.

The equivalents between the 8080 flags and the flag register in the 8086 are summarized in Figure 3.4. The low-end 8 bits are treated as 8080 and 8086 flags, while the high-end 8 bits are new flag bits associated with the 8086.

The 8086 segment registers are shown in Figure 3.5. These 16-bit registers are defined as follows:

CS: code segment.

DS: data segment.

SS: stack segment.

ES: extra segment.

The segment registers are used in memory address determinations. All instruction fetches are taken by the 8086 as relative to the current contents of the CS register, using the instruction pointer (IP) as an offset.

All data references (with a few exceptions) are taken to be relative to the current contents of the DS register. The exceptions are those data references involving BP or SP, or DI in a string instruction.

All data references that involve SP or BP are taken by default to be relative to SS. Examples include push and pop instructions or call and return operations.

Finally, data references in string instructions that use DI are taken to be relative to ES, the current extra segment.

The registers in the 8086 are summarized in Figure 3.6, which also shows the equivalent 8080 registers shaded. As we point out above, since the 8080 registers are a subset of the 8086 registers, programs written for the 8080 can be transferred over to the 8086, using the subset of registers shown in Figure 3.6, without much difficulty.

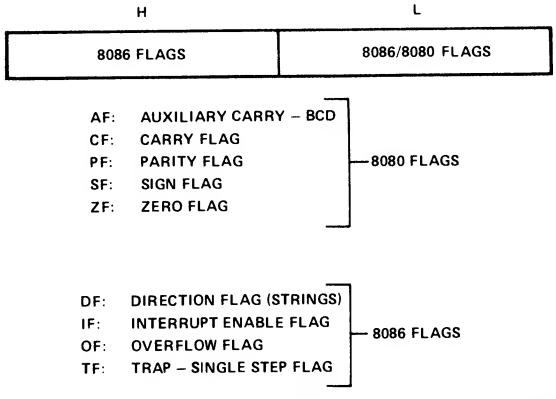


FIGURE 3.4 Intel 8086 and 8080 flag register equivalents. (Source: Intel Corporation.)

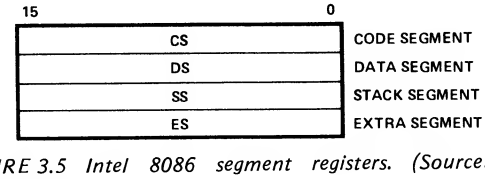


FIGURE 3.5 Intel 8086 segment registers. (Source: Intel Corporation.)

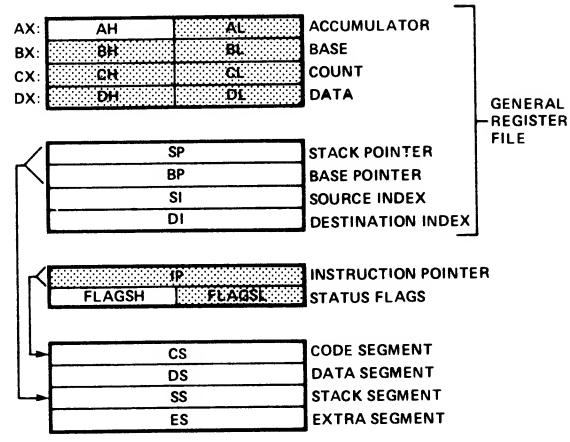


FIGURE 3.6 Intel 8086 registers (showing 8080 register equivalents shaded). (Source: Intel Corporation.)

System Configuration

System configuration is concerned with the means of interfacing memory and peripherals to the 8086. Depending on the number and complexity of the peripheral devices, one can choose one of two basic system configuration modes available with the Intel 8086—maximum and minimum. These two modes are defined by a pin (MN/MX) on the processor that is strapped either to ground or V_{cc} potential. The definition of certain of the pins of the processor is dependent on potential level to which the MN/MX pin is strapped. When the MN/MX pin is strapped to ground, the processor is in a maximum mode configuration; when it is strapped to V_{cc}, the processor is in the minimum mode configuration.

An example of the minimum mode configuration is shown in Figure 3.7. In the minimum mode, the 8086 itself generates the control signals used by the memory

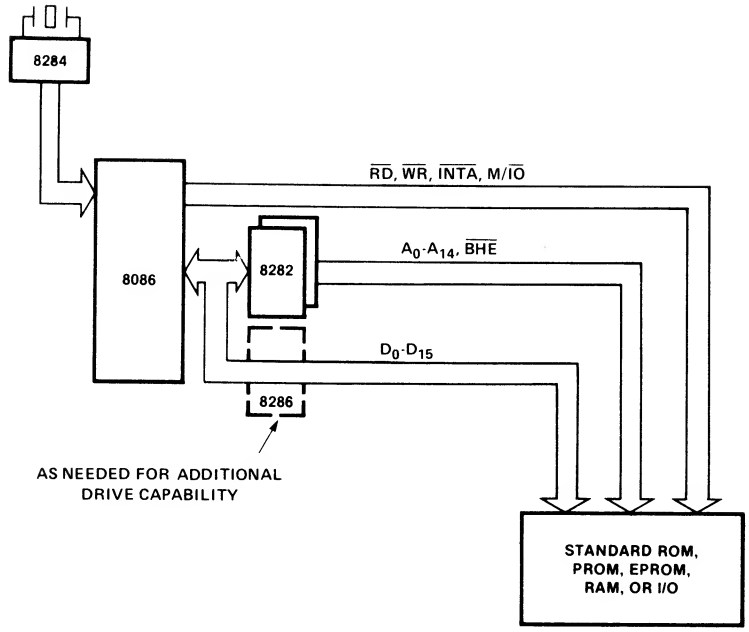


FIGURE 3.7 Intel 8086 minimum mode system configuration. (Source: Intel Corporation.)

and I/O devices, as well as a timing signal for latching the addresses during the T_1 state. (The timing states of the 8086 are described subsequently.)

An example of the maximum mode configuration is shown in Figure 3.8. In the maximum mode, an 8288 bus controller is used for generating the control signals to memory and I/O. This frees the minimum mode control signal pins to take on additional functions, such as extra direct memory access control and bus-locking capabilities.

In the minimum mode, the definitions of the pins 24–31 are as follows:

24 INTA
25 ALE
26 DEN
27 DT/R
28 M/IO
29 WR
30 HLDA
31 HOLD

The definitions of these minimum mode signals are given below:

- INTA is a read strobe for interrupt acknowledge.
- ALE is an address latch enable.
- DEN is data enable.
- DT/R is data transmit/receive.
- M/IO is used to distinguish a memory access from an I/O access (M = HIGH, IO = LOW).
- WR is write strobe, which indicates that the process is performing a write memory or write IO/O cycle.
- HOLD indicates that another bus master is requesting a local bus hold.

A more detailed example of a minimum mode implementation is shown in Figure 3.9.

In maximum mode, the definitions of pins 24–31 are as follows:

24 QS1
25 QS0
26 $\overline{S_0}$
27 $\overline{S_1}$
28 $\overline{S_2}$
29 $\overline{LOCK}$
30 $\overline{RQ}/\overline{GT1}$
31 $\overline{RQ}/\overline{GT0}$

The definition of these maximum mode signals are given below:

$\overline{S_2}$	$\overline{S_1}$	$\overline{S_0}$	
0 (LOW)	0	0	Interrupt acknowledge
0	0	1	Read I/O port
0	1	0	Write I/O port
0	1	1	Halt
1 (HIGH)	0	0	Code access
1	0	1	Read memory
1	1	0	Write memory
1	1	1	Passive

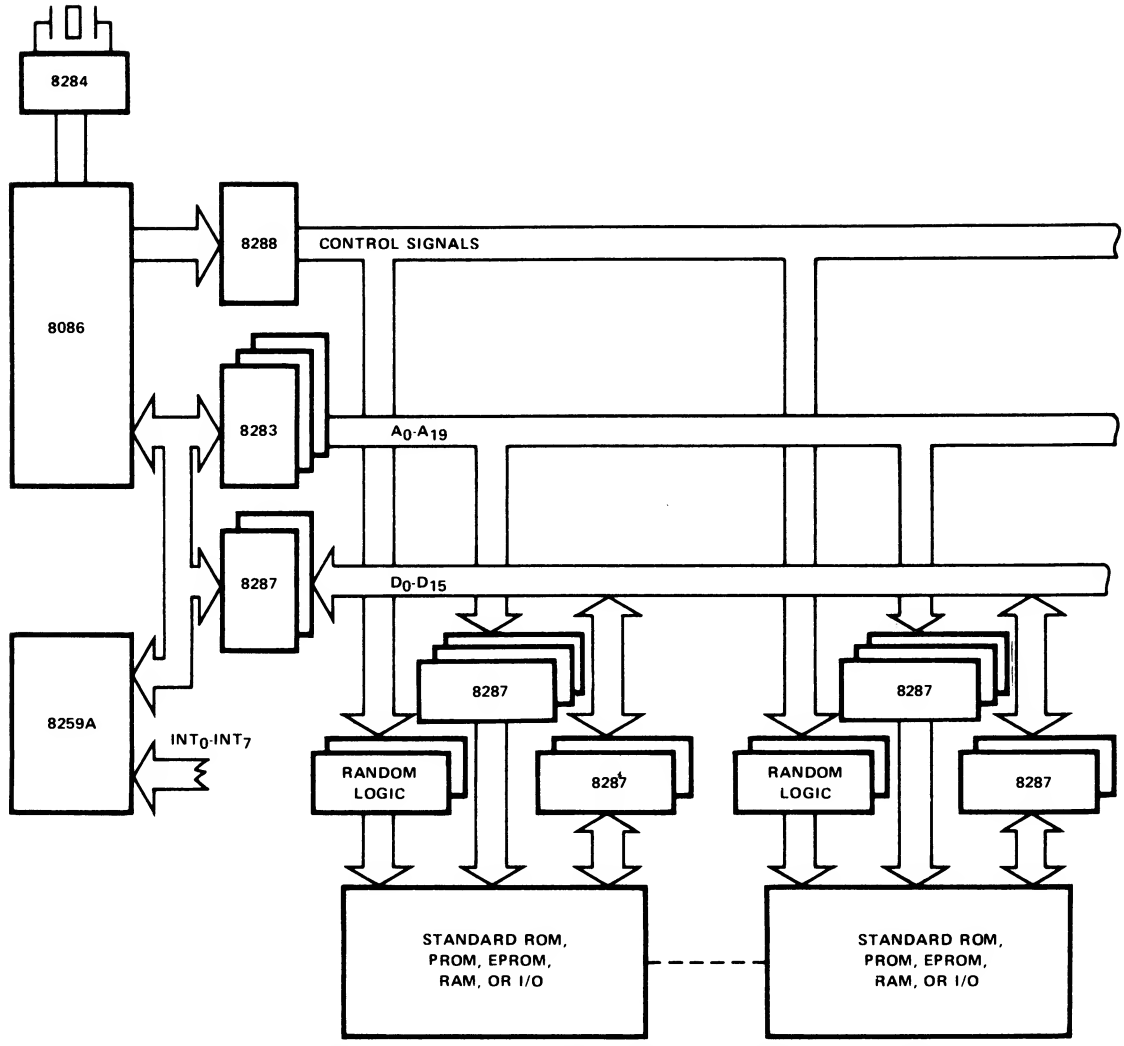


FIGURE 3.8 Intel 8086 maximum mode system configuration. (Source: Intel Corporation.)

QS_1	QS_0	
0 (LOW)	0	No operation
0	1	First byte of op code from queue
1 (HIGH)	0	Empty the queue
1	1	Subsequent byte from queue

"$\overline{S}_2, \overline{S}_1, \overline{S}_0$ (Output 3-State)

"These status lines are encoded as follows:
Status is active during T_4, T_1, and T_2 and is returned to the passive state (1,1,1) during T_3 or during T_W when READY is HIGH. This status is used by the 8288 Bus Controller to generate all memory and I/O access control signals. Any change by $\overline{S_2}$, $\overline{S_1}$, or $\overline{S_0}$ during T_4 is used to indicate the beginning of a bus cycle, and the return to the passive state in T_3 or T_W is used to indicate the end of a bus cycle.

"These signals float to 3-state OFF in "hold acknowledge."

$\overline{RQ}/\overline{GT}_0$, $\overline{RQ}/\overline{GT}_1$ (Input/Output)

"The request/grant pins are used by other local bus masters to force the processor to release the local bus at the end of the processor's current bus cycle. Each pin is bidirectional with $\overline{RQ}/\overline{GT}_0$ having higher priority than $\overline{RQ}/\overline{GT}_1$. $\overline{RQ}/\overline{GT}$ has an internal pull-up resistor so may be left unconnected. The request/grant sequence is as follows:

1. A pulse of 1 CLK wide from another local bus master indicates a local bus request ("hold") to the 8086 (pulse 1).
2. During the CPU's next T_4 or T_1 a pulse 1 CLK wide, from the 8086 to the requesting master (pulse 2), indicates that the 8086 has allowed the local bus

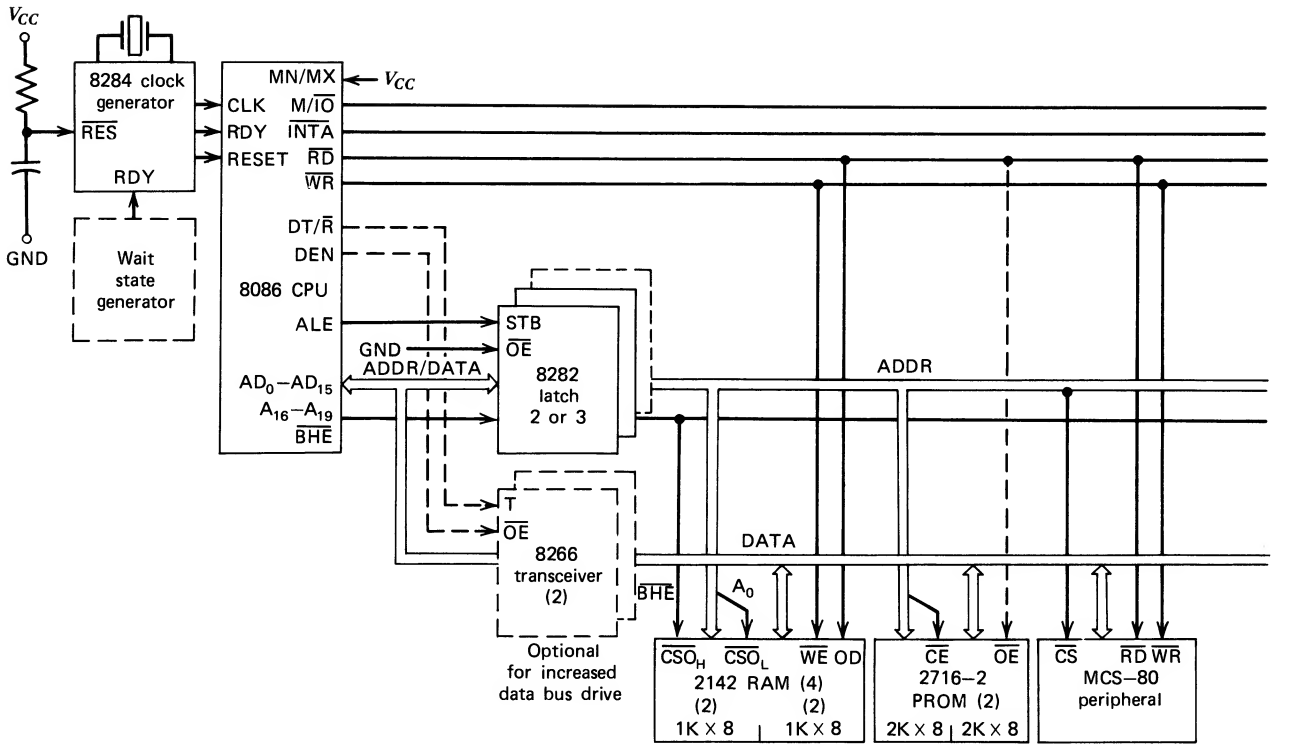


FIGURE 3.9 Intel 8086 minimum mode example. (Source: Intel Corporation.)

to float and that it will enter the "hold acknowledge" state at the next CLK. The CPU's bus interface unit is disconnected logically from the local bus during "hold acknowledge."

3. A pulse 1 CLK wide from the requesting master indicates to the 8086 (pulse 3) that the "hold" request is about to end and that the 8086 can reclaim the local bus at the next CLK. The CPU then enters T_4.

Each master-master exchange of the local bus is a sequence of 3 pulses. There must be one dead CLK cycle after each bus exchange. Pulses are active LOW.

"$\overline{\text{LOCK}}$"

"The $\overline{\text{LOCK}}$ output indicates that other system bus masters are not to gain control of the system bus while $\overline{\text{LOCK}}$ is active LOW. The $\overline{\text{LOCK}}$ signal is activated by the "LOCK" prefix instruction and remains active until the next instruction is extracted from the queue. This signal is active LOW, and floats to 3-state OFF in "hold acknowledge."

"QS_1, QS_0"

"QS_1 and QS_0 provide status to allow external tracking of the internal 8086 instruction queue. The queue status is valid during the CLK cycle after which the queue operation is performed.*"

A more detailed example of a typical maximum mode configuration is shown in Figure 3.10. In such a system, use is made of the 8288 bus controller, an 8282/8283 latch for latching the system addresses, and an 8286/8287 transceiver for bus loading.

The Intel 8288 bus controller is a 20-pin bipolar component for use with medium to large 8086 systems. A functional block diagram of the 8288 is shown in Figure 3.11, together with the pin configuration of the component.

System Timing

The system timing for the 8086 is represented by the timing diagram in Figure 3.12. The first signal represented is the clock, labeled CLK. Each processor bus cycle consists of at least four CLK cycles, which are sequentially labeled T_1, T_2, T_3 and T_4. One or more wait states, labeled T_{wait} may be inserted between the T_3 and T_4 cycles.

During the T_1 cycle, the address latch enable (ALE) signal is emitted. Once the external circuitry receives the

**SOURCE: Intel Corporation. Reprinted by permission.*

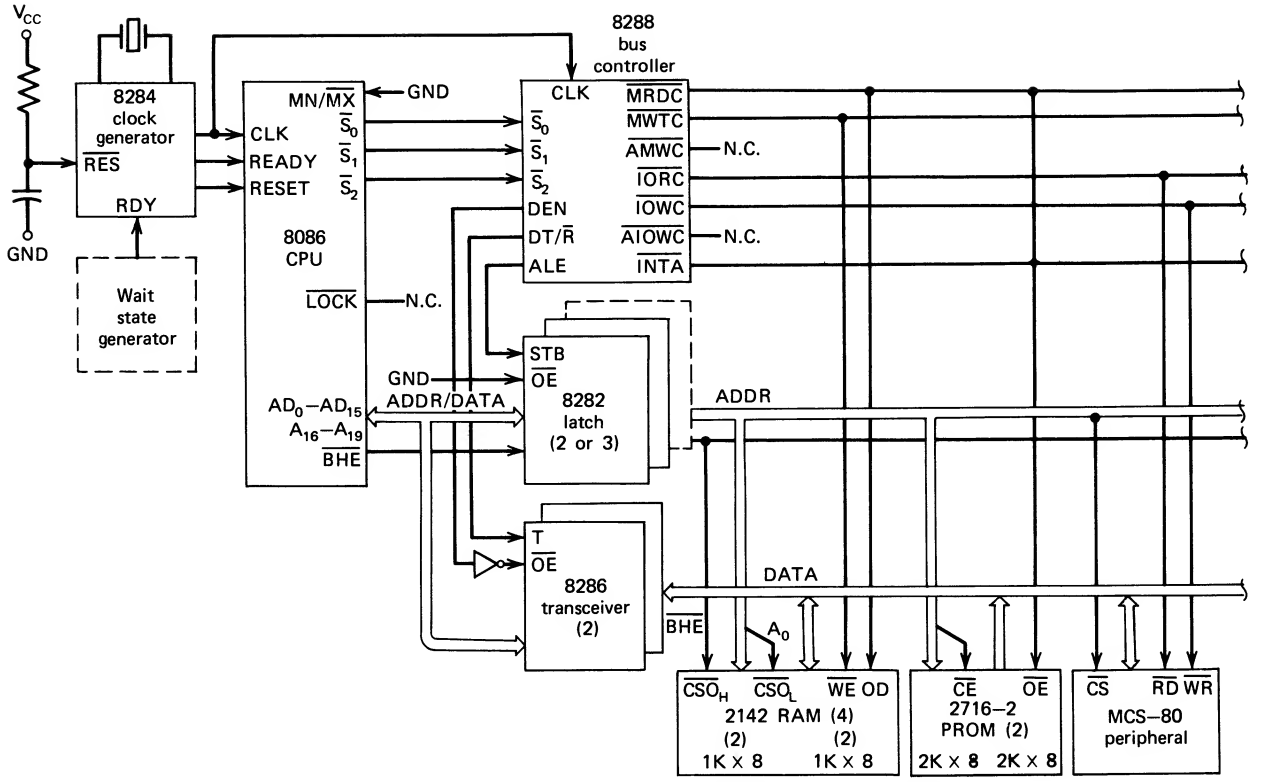


FIGURE 3.10 Intel 8086 maximum mode example. (Source: Intel Corporation.)

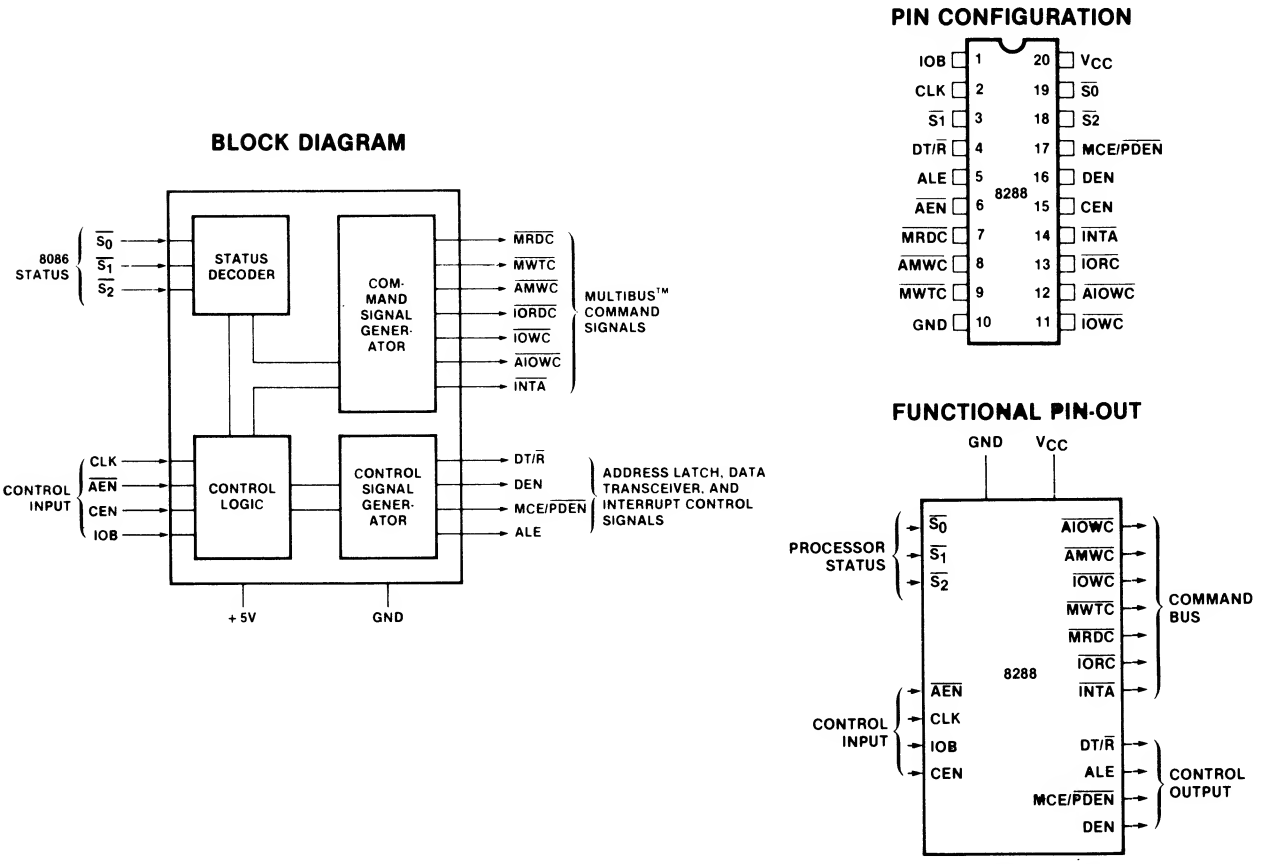


FIGURE 3.11 Intel 8288 functional block diagram and pin configuration. (Source: Intel Corporation.)

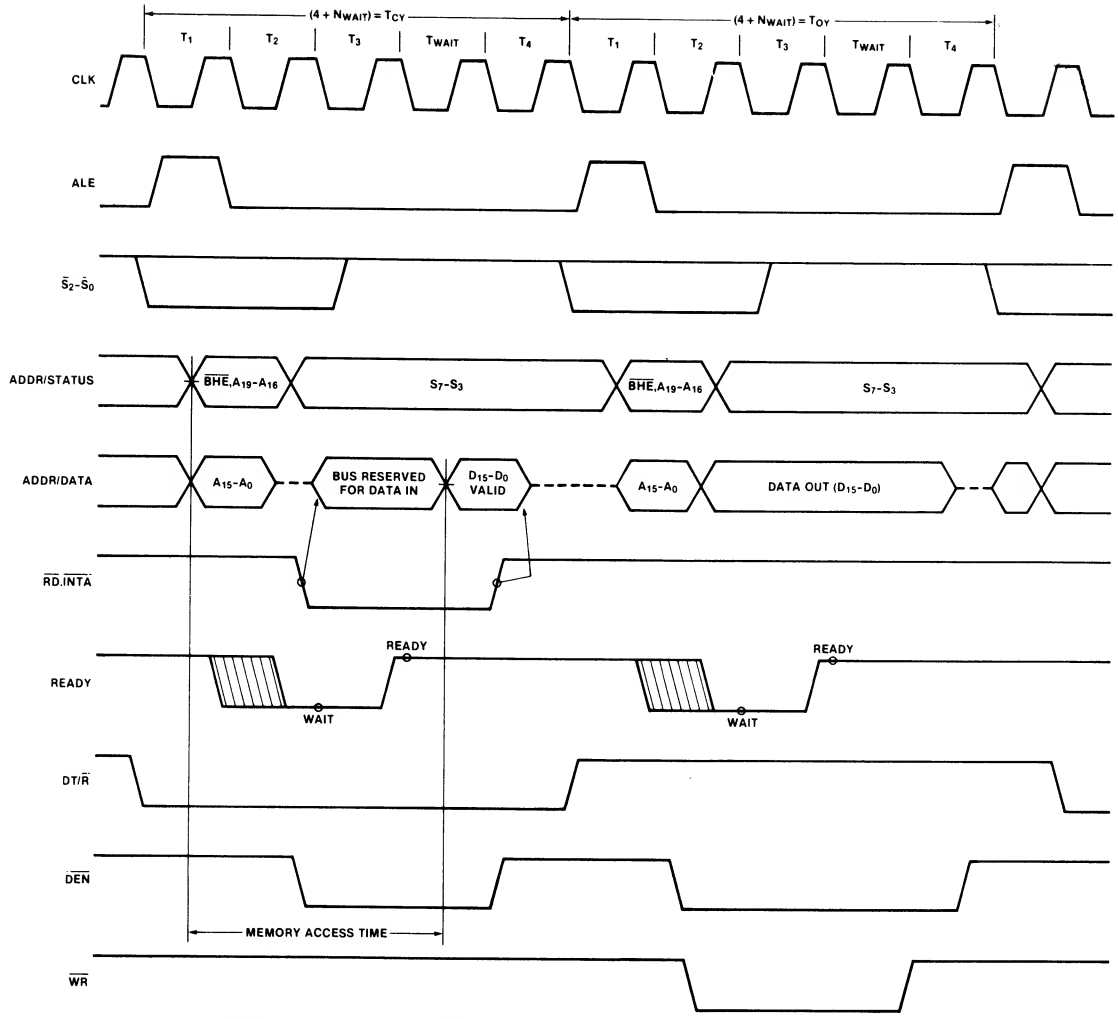


FIGURE 3.12 Intel 8086 timing diagram. (Source: Intel Corporation.)

trailing edge of this pulse, information concerning address and status information may be latched.

During the T_1 cycle, status information is also supplied over the S_0, S_1, and S_2 pins, which is used to specify the type of bus transaction taking place. Such determination is made according to the following

	$\overline{S_2}$	$\overline{S_1}$	$\overline{S_0}$
Interrupt acknowledge	0	0	0
Read I/O	0	0	1
Write I/O	0	1	0
Halt	0	1	1
Instruction fetch	1	0	0
Read data from memory	1	0	1
Write data to memory	1	1	0
No bus cycle	1	1	1

The timing diagram indicates that the status lines S_2 - S_0 take values of either 0 or 1 during the appropriate states of the timing cycle.

Further status information is provided by the signals S_3 - S_7. These status signals are multiplexed together with the high-order address bits and therefore appear during the T_2 - T_4 states. The S_3 and S_4 bits are used to specify which of the segment registers are used to form the address. The combinations are as follows:

S_4	S_3	*Abbreviation*	*Name*
0	0	ES	Extra data segment
0	1	CS	Code segment
1	0	SS	Stack segment
1	1	DS	Data segment

The memory may therefore be divided into four segments of up to 64K bytes each, with each segment falling within 16-byte boundaries, as shown in Figure 3.13. The external logic can therefore decode these status signals and access the appropriate portion of the memory.

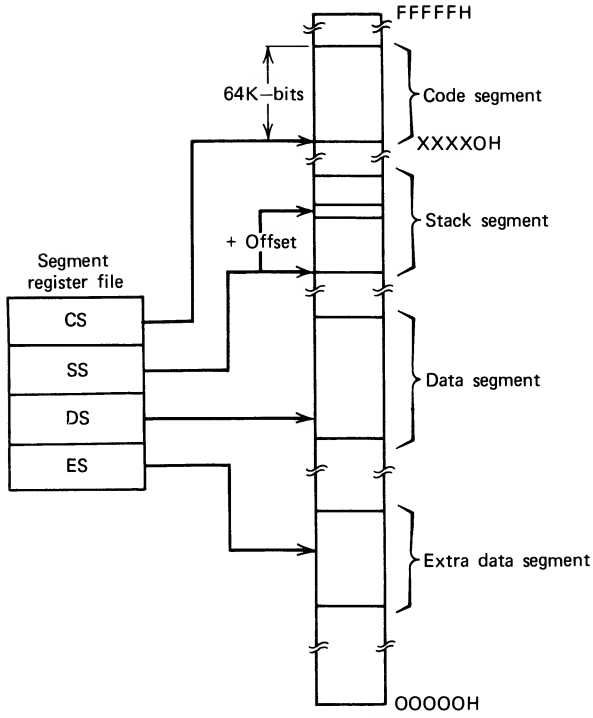


FIGURE 3.13 Intel 8086 memory segmentation. (Source: Intel Corporation.)

The remaining status signals S_5, S_6, and S_7 are not of concern to the user at this point.

The timing diagram also indicates the BHE signal (bus high enable) is present during the T_1 cycle. The BHE signal is used to enable data onto the data pins D_{15}-D_8.

Continuing down the timing diagram in Figure 3.12, the next signals shown are the address and data signals, ADDR/DATA, using pins AD_{15}-AD_0. The address is supplied during the T_1 cycle, and data (either in or out) are supplied during T_2-T_4.

The next signal line timing shown is the $\overline{RD}$.$\overline{INTA}$. $\overline{RD}$ is a read strobe output that indicates the processor is performing a memory or I/O read cycle. $\overline{RD}$ is low during T_2, T_3, and T_W. It is a three-state signal, and floats to three-state OFF during "hold acknowledge." $\overline{INTA}$ is a similar read strobe that is active during T_2, T_3, and T_W of each interrupt acknowledge cycle. Like $\overline{RD}$, it floats to the three-state OFF during hold acknowledge.

Continuing down the timing diagram of Figure 3.12, the READY signal is an acknowledgment from the addressed memory or the I/O device that it will complete the data transfer. It is active in the HIGH state.

The next signal line is DT/R, or data transmit/receive. This signal is used in the minimum system configuration to control the direction of data flow through the 8286 transceiver.

DEN and WR, two other signals that are characteristic of the minimum mode, are also shown in the timing diagram of Figure 3.12.

DEN is data enable and is active LOW during each memory or I/O access and for interrupt cycles.

WR is write strobe and is active LOW during a write memory or write I/O cycle.

The signal definitions of a number of other pins are given below.

- INTR is the interrupt request and is an active HIGH signal. This input is sampled during the last clock cycle of each instruction. If the INTR pin is high, an interrupt vector lookup table is referenced, and the processor vectors to a subroutine.
- TEST is an input pin that is examined by the "wait for test" operation. If the TEST input is low, execution continues; otherwise the processor waits in an idle state.
- NMI is nonmaskable interrupt and is an input that causes a different type of interrupt than INTR. Whereas INTR can be internally masked by software, MNMI cannot be masked.
- RESET is an input that causes the processor to stop what it has been doing and wait until the RESET pin returns to LOW.

Consideration should also be given to system timing during during read, write, and interrupt cycles. Figure 3.14 shows the timing for such operations in a minimum mode configuration, and Figure 3.15 shows the timing in a maximum mode configuration.

Instruction Set

The instruction set of the 8086 is divided into six categories:

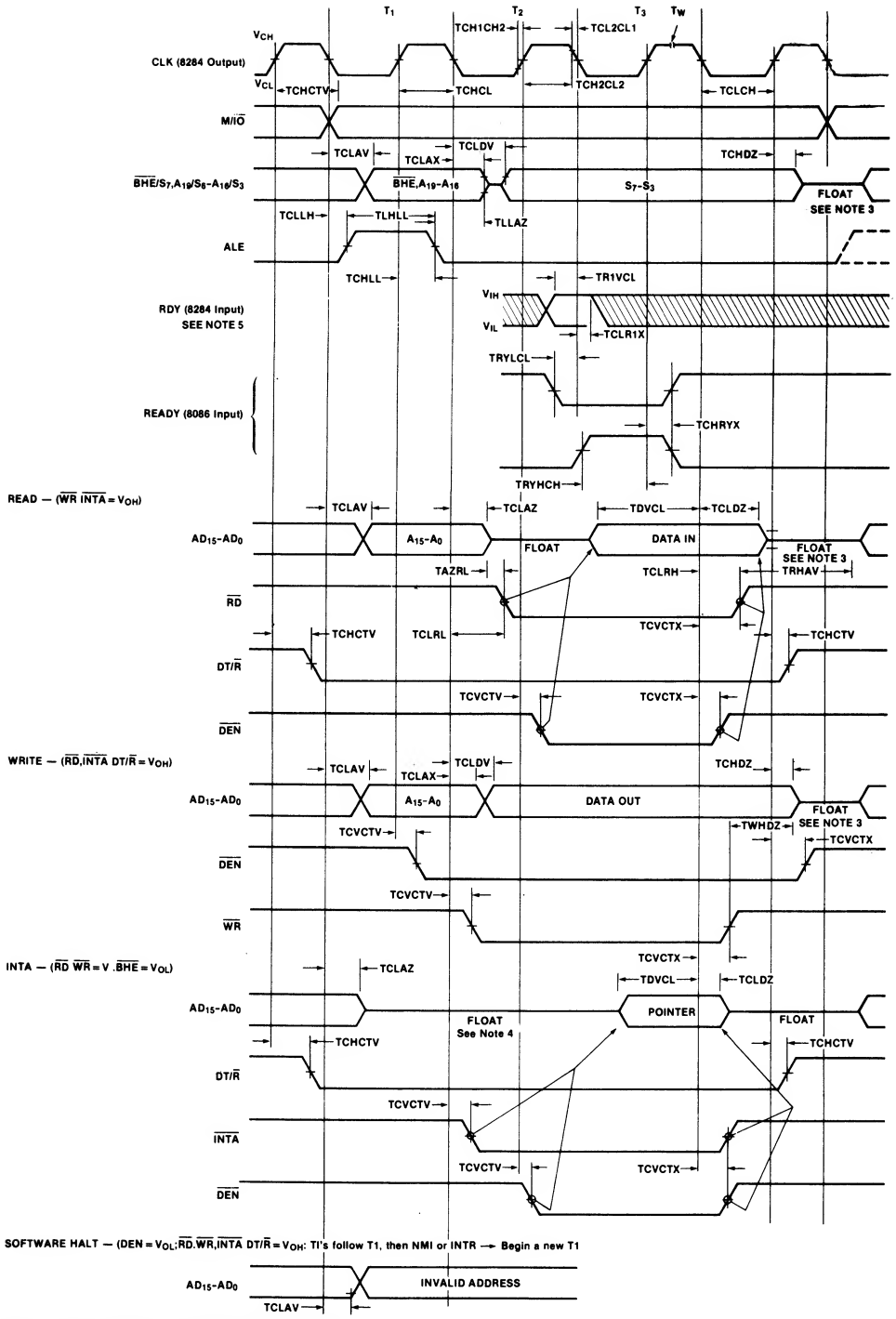
- Data transfer.
- Arithmetic.
- Logic.
- String manipulation.
- Control Transfer.
- Processor control.

We consider each of these categories in turn, as well as representative instructions in each category.

Data Transfer

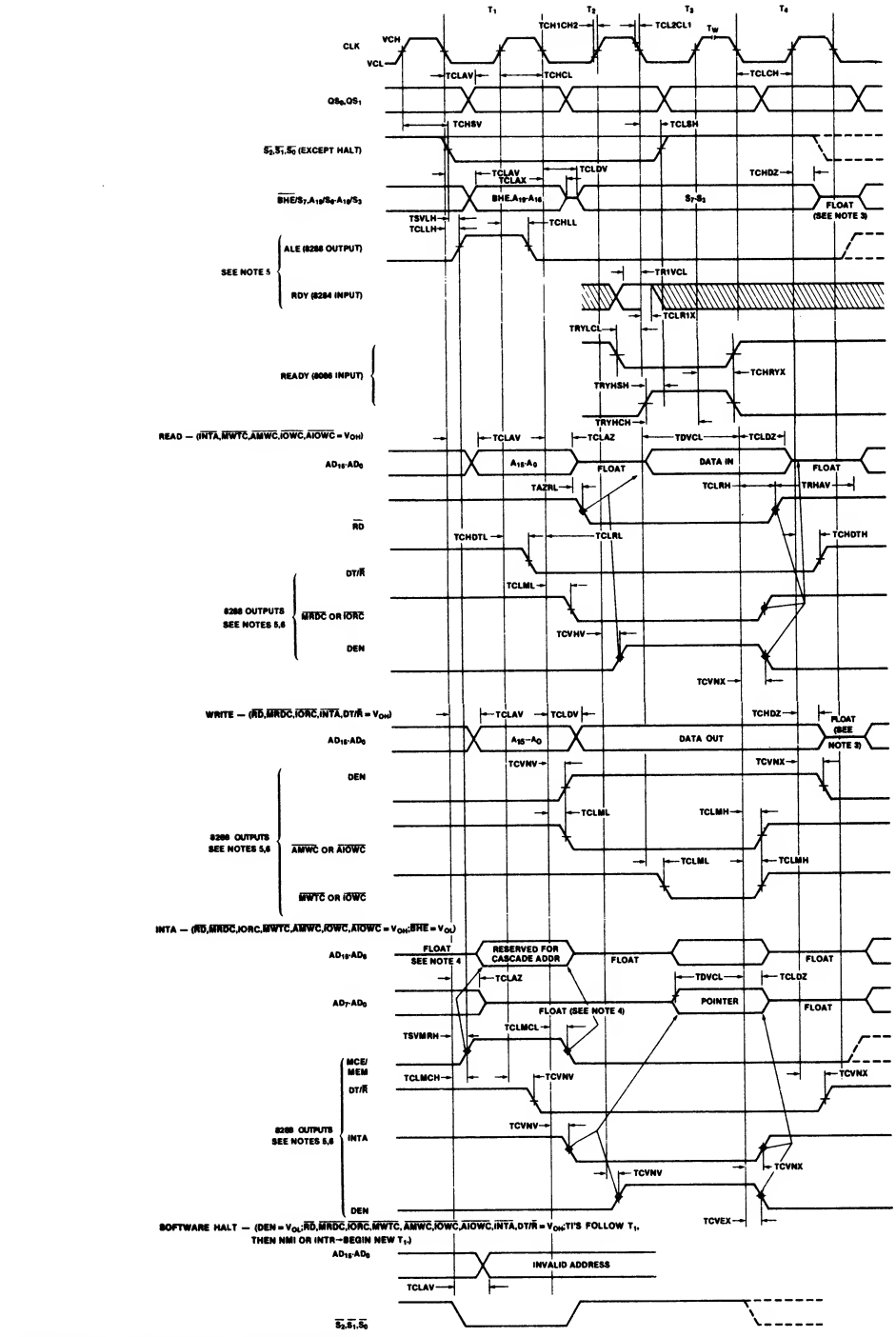
There are six basic data transfer instructions:

MOV	Move
PUSH	Push



NOTES:
1. ALL SIGNALS SWITCH BETWEEN V_{OH} AND V_{OL} UNLESS OTHERWISE SPECIFIED.
2. RDY IS SAMPLED NEAR THE END OF T_2,T_3,T_W TO DETERMINE IF T_W MACHINES STATES ARE TO BE INSERTED.
3. FOLLOWING A WRITE CYCLE DATA REMAINS VALID ON THE 8086 LOCAL BUS UNTIL A LOCAL BUS MASTER DECIDES TO RUN ANOTHER BUS CYCLE. THE LOCAL BUS IS FLOATED BY THE 8086 ONLY WHEN THE 8086 ENTERS A "HOLD ACKNOWLEDGE" STATE.
4. TWO INTA CYCLES RUN BACK-TO-BACK. THE 8086 LOCAL ADDR/DATA BUS IS FLOATING DURING THE SECOND INTA CYCLE.
5. SIGNALS AT 8284 OR 8286 ARE SHOWN FOR REFERENCE ONLY.
6. ALL TIMING MEASUREMENTS ARE MADE AT 1.5V UNLESS OTHERWISE NOTED.

FIGURE 3.14 Intel 8086 bus timing, minimum mode. (Source: Intel Corporation.)



NOTES:
1. ALL SIGNALS SWITCH BETWEEN V_{OH} AND V_{OL} UNLESS OTHERWISE SPECIFIED.
2. RDY IS SAMPLED NEAR THE END OF T_2,T_3,T_W TO DETERMINE IF T_W MACHINES STATES ARE TO BE INSERTED.
3. FOLLOWING A WRITE CYCLE DATA REMAINS VALID ON THE 8086 LOCAL BUS UNTIL A LOCAL BUS MASTER DECIDES TO RUN ANOTHER BUS CYCLE. THE LOCAL BUS IS FLOATED BY THE 8086 ONLY WHEN THE 8086 ENTERS A "HOLD ACKNOWLEDGE" STATE.
4. TWO INTA CYCLES RUN BACK-TO-BACK. THE 8086 LOCAL ADDR/DATA BUS IS FLOATING DURING THE SECOND INTA CYCLE.
5. SIGNALS AT 8284 OR 8288 ARE SHOWN FOR REFERENCE ONLY.
6. THE ISSUANCE OF THE 8288 COMMAND AND CONTROL SIGNALS (MRDC,MWTC,AMWC,IORC,IOWC,AIOWC,INTA, AND DEN) LAGS THE ACTIVE HIGH 8288 CEN.
7. ALL TIMING MEASUREMENTS ARE MADE AT 1.5V UNLESS OTHERWISE NOTED

FIGURE 3.15 Intel 8086 bus timing, maximum mode. (Source: Intel Corporation.)

POP	Pop
XCHG	Exchange
IN/INW	Input to AL/AX
OUT/OUTW	Output from AL/AX

Arithmetic

There are seven basic arithmetic instructions:

ADD	Add
ADC	Add with carry
INC	Increment
SUB	Subtract
SBB	Subtract with borrow
DEC	Decrement
CMP	Compare

In addition, there are also signed and unsigned multiply and divide instructions.

Logic

NOT	Invert
SHR	Shift logical right
SAR	Shift arithmetic right
AND	"And"
TEST	"And" function to flags
OR	"Or"
XOR	Exclusive or

String Manipulation

MOVB	Move byte
CMPB	Compare byte
SCAB	Scan byte
LODB	Load byte to AL register

Control Transfer

CALL	Call
JMP	Unconditional jump
RET	Return from CALL
INT	Interrupt

Processor Control

HLT	Halt
WAIT	Wait

Memory Addressing

The memory in an 8086 system is made up of a sequence of 8-bit bytes, and any two consecutive bytes

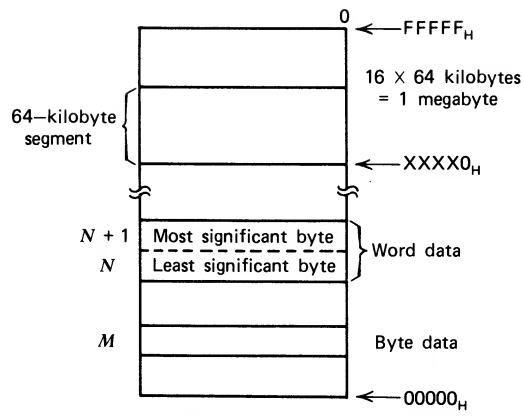


FIGURE 3.16 Intel 8086 memory organization. (Source: Intel Corporation.)

form a 16-bit word. The entire memory itself is broken up into an arbitrary number of segments, each consisting of 64K successive bytes, such as suggested in Figure 3.16.

The concept of segments is important since it is through such segments that memory addressing takes place. Each segment must begin at an address that is evenly divisible by 16, that is, the low-order 4 bits are 0 0 0 0. The 8086 performs memory addressing through such segments.

It should be recalled that the 8086 is a 16-bit processor, and therefore the addresses it manipulates are 16-bits long. A 16-bit address can only address a maximum of 64K bytes. Therefore, once a given 64K segment has been specified, the 8086 can directly and immediately address any byte within that segment.

The method of sepcifying the segment of interest is through the segment registers that we noted earlier in Figure 3.5 and 3.13. Since there are four segment registers, at any given moment the 8086 can directly address the contents of the four specified segments. These four directly addressable segments are called:

- Code segment.
- Data segment.
- Stack segment.
- Extra data segment.

The only requirement for the definition of a segment is that the low-order 4 bits of a segment address be 0 0 0 0. Segments may overlap one another.

The four segment registers of the 8086 that provide direct and immediate addressability of four segments as suggested in Figure 3.13, are:

- Current code segment.
- Current data segment.

- Current stack segment.
- Current extra data segment.

These segments may also overlap and need not be unique.

To address bytes or words within a segment, a 16-bit offset address or "effective address" (EA) is used, together with the contents of one of the segment registers. The programmer has a capability of specifying which segment register is to be used in addressing, and the designation of that segment register is incorporated within the instruction itself.

As we point out in Chapter 2, the 8086 has the capability of addressing up to 1 megabyte of memory (i.e., 2^{20} = 1,048,576 bytes). The 20-bit physical address is constructed by the 8086 using both the offset address or EA provided by the user with the instruction and the content of the designated segment register. This construction begins by shifting the segment address left four places, and 0 0 0 0 is appended at the four low-order bit positions. The effective address is then added to this 20-bit number to form the actual physical address. This construction process is shown in Figure 3.17.

We point out above that the programmer has the capability of specifying the segment register to be used, in general. The one exception is the code segment register CS, which always must be used when a code address is generated and cannot be overridden. When a data address is generated, DS is generally used, and when a stack address is generated SS is generally used. The use of DS and SS can, however, be overridden by the programmer by preceding the instruction with a special 1-byte segment override prefix. When such a prefix is used, a specific segment register is used for all data references during the execution of that instruction. The use of such segment overrides is summarized in Table 3.1.

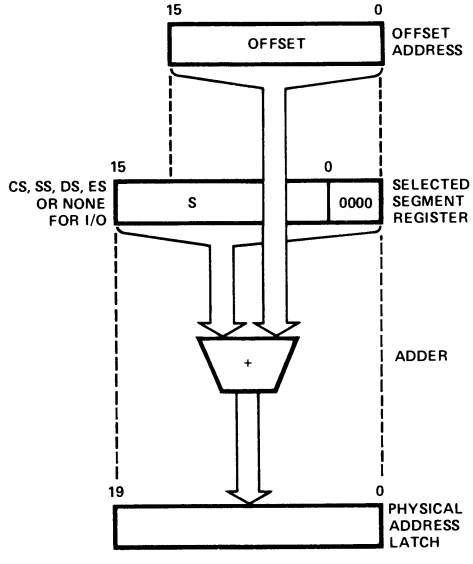


FIGURE 3.17 Intel 8086 memory addressing. (Source: Intel Corporation.)

TABLE 3.1. Segment Overrides

Default Condition	*With Override Prefix*
IP + CS = code address	No override
SP + SS = stack address	No override
BP + SS = stack address	BP + DS
	ES
or stack marker	DS
EA + DS = data address	EA + ES
	SS
	CS

The basic form of the 8086 instructions is shown in Figure 3.18. It is seen that the first byte comprises an op code, a 1-bit D field, and a 1-bit W field. The D field identifies direction: if D = 1, then it is a direction "to"; if D = 0, then the instruction refers to a direction "from." The W field is used to distinguish byte instructions (1 byte = 8 bits) from word instructions (1 word = 16 bits): if W = 1, then it is a word instruction; if W = 0, then it is a byte instruction.

For example, the first byte of the MOVE instruction is

1 0 0 0 1 0 d w

There are four possibilities. The first,

1 0 0 0 1 0 0 0

is a byte transfer to the memory or register operand *from* the register operand. The second possibility,

1 0 0 0 1 0 1 0

is a byte transfer from the memory or register operand *to* the register operand. The third possibility,

1 0 0 0 1 0 0 1

is a word transfer to the memory or register operand from the register operand. Finally, the fourth possibility,

1 0 0 0 1 0 1 1

is a word transfer from the memory or register operand to the register operand.

In all these possibilities, the questions of whether it is a memory or register operand in the first case and *which* register in the second case must be answered by the second byte of the instruction.

As shown in Figure 3.18, the second byte of the instruction is divided into MOD, REG, and R/M fields.

The MOD or mode field refers to the addressing mode being used. In particular, it specifies how the data in the R/M field are to be used in finding the operand.

If MOD = 1 1, then the R/M field is treated as a register field. The definition of the registers of the 8086 in terms of a 3-bit register field is given in Table 3.2.

The REG or register field is used to specify either an 8-bit register or a 16-bit register, which is a location of the operand. The particular registers specified by the field are defined in Table 3.2.

The R/M or register/memory field specifies either the location of the operand or the location in memory where the operand can be found. One or two bytes may follow, which is the address displacement if a R/M field requires such. Table 3.3 indicates how the effective address (EA) of the memory operand is computed based upon the R/M field. DISP refers to the displacement byte that follows the R/M field.

The effective address (EA), we recall from the section on memory addressing above, is used together with the contents of the segment register to construct the actual physical address. The effective address is only equal to the actual physical address in memory when the contents of the segment register are 0.

In the section that follows, we present a detailed listing of the entire instruction set of the 8086, taken directly from Intel Corporation publications. This listing presents:

- The mnemonic for the instruction, such as for the Intel ASM86 assembler.
- The encoding.
- The source (SRC) and destination (DEST) operands.
- Time required for execution (in clock cycles).

Each clock cycle is 200 nsec if the 5-MHz clock is used in the 8086 system, or 125 nsec if the 8-MHz clock is used. The number of clock cycles per instruction is fixed if the instruction utilizes register or immediate operands, but is variable depending on the addressing mode if the instruction uses a memory operand. Table 3.4 gives this variable time as "EA," which is added to some fixed number of clock cycles depending on the instruction.

TABLE 3.2. Designation of Register Fields Word Operation (W = 1)

Word Operation (W = 1)	
0 0 0	AX
0 0 1	CX
0 1 0	DX
0 1 1	BX
1 0 0	SP
1 0 1	BP
1 1 0	SI
1 1 1	DI
Byte Operation (W = 0)	
0 0 0	AL
0 0 1	CL
0 1 0	DL
0 1 1	BL
1 0 0	AH
1 0 1	CH
1 1 0	DH
1 1 1	BH

TABLE 3.3. R/M Field Definition

R/M	*Effective Address (EA)*
0 0 0	BX + SI + DISP
0 0 1	BX + DI + DISP
0 1 0	BP + SI + DISP
0 1 1	BP + DI + DISP
1 0 0	SI + DISP
1 0 1	DI + DISP
1 1 0	BP + DISP
1 1 1	BX + DISP

TABLE 3.4. Memory Access Timing

Memory Addressing Mode	*Additional Clock Cycles ("EA")*
Direct 16-bit offset address	6
Indirect through base or index register	5
Indirect through base or index register with displacement constant	9
Indirect through sum of index-register plus base-register	7 or 8
Indirect through sum of base register plus index register with displacement constant	11 or 12

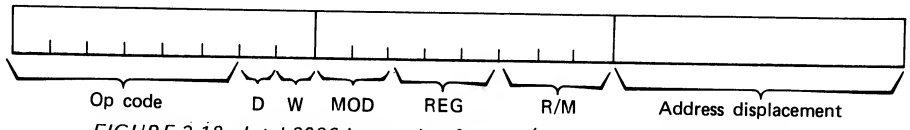


FIGURE 3.18 Intel 8086 instruction format. (Source: Intel Corporation.)

8086 Instruction Set

Data Transfer

Four classes of data transfer operations may be distinguished: general purpose, accumulator specific, address-object transfers, and flag transfers. None of the transfer instructions affect any flag settings except SAHF and POPF.

Four general purpose data transfer operations are provided. These may generally be applied to most operands, though there are specific exceptions. The general purpose transfers (except XCHG) are the only operations which allow a segment register as an operand:

MOV Move
PUSH Push
POP Pop
XCHG Exchange

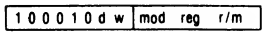
INSTRUCTION SET

Mnemonic: **MOV**

Description: MOV performs a byte or word transfer from the source operand to the destination operand.

Encoding:

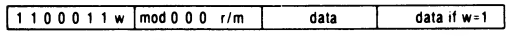
Memory or Register Operand to/from Register Operand:



if d = 1 then SRC = EA, DEST = REG
else SRC = REG, DEST = EA

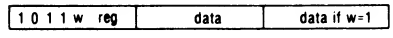
Timing (clocks): register to register 2
memory to register 8+EA
register to memory 9+EA

Immediate Operand to Memory or Register Operand:



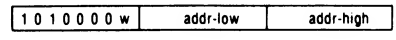
SRC = data, DEST = EA
Timing: 10+EA clocks

Immediate Operand to Register:



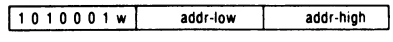
SRC = data, DEST = REG
Timing: 4 clocks

Memory Operand to Accumulator:



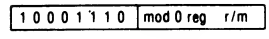
if w = 0 then SRC = addr, DEST = AL
else SRC = addr+1:addr, DEST = AX
Timing: 10 clocks

Accumulator to Memory Operand:



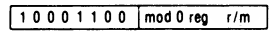
if w = 0 then SRC = AL, DEST = addr
else SRC = AX, DEST = addr+1:addr
Timing: 10 clocks

Memory or Register Operand to Segment Register:



if reg ≠ 01 then SRC = EA, DEST = REG
else undefined operation
Timing (clocks): register to register 2
memory to register 8+EA

Segment Register to Memory or Register Operand:



SRC = REG, DEST = EA

Timing (clocks): memory to register 9+EA
register to register 2

Operation:

(DEST) <== (SRC)

Flags Affected:

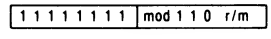
None

Mnemonic: **PUSH**

Description: PUSH decrements the SP register by two and then transfers a word from the source operand to the stack element currently addressed by SP.

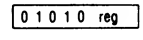
Encoding:

Memory or Register Operand:



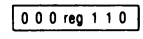
SRC = EA
Timing (clocks): memory 16+EA
register 10

Register Operand:



SRC = REG
Timing: 10 clocks

Segment Register:



SRC = REG
Timing: 10 clocks

Operation:

(SP) <== (SP) - 2
((SP)+1:(SP)) <== (SRC)

Flags Affected:

None

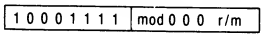
From *MCS-86 User's Manual.* Mnemonics © Intel, 1978. Reprinted by permission of Intel Corporation.

Mnemonic: **POP**

Description: POP transfers a word operand from the stack element addressed by the SP register to the destination operand and then increments SP by 2.

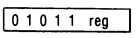
Encoding:

Memory or Register Operand:



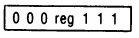
DEST = EA
Timing (clocks): memory 17+EA
register 8

Register Operand:



DEST = REG
Timing: 8 clocks

Segment Register:



if reg ≠ 01 then DEST = REG
else undefined operation
Timing: 8 clocks

Operation:

(DEST) <== ((SP)+1:(SP))
(SP) <== (SP)+2

Flags Affected:

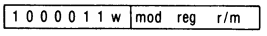
None

Mnemonic: **XCHG**

Description: XCHG exchanges the byte or word source operand with the destination operand. The segment registers may not be operands of XCHG.

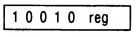
Encoding:

Memory or Register Operand with Register Operand:



SRC = EA, DEST = REG
Timing (clocks): memory with register 17+EA
register with register 4

Register Operand with Accumulator:



SRC = REG, DEST = AX
Timing: 3 clocks

Operation:

(Temp) <== (DEST)
(DEST) <== (SRC)
(SRC) <== (Temp)

Flags Affected:

None

Three Accumulator transfer operations are provided:

Accumulator Specific Transfers

IN	Input Byte	Single Operation
INW	Input Word	
OUT	Output Byte	Single Operation
OUTW	Output Word	
XLAT	Translate	

From *MCS-86 User's Manual.* Mnemonics © Intel, 1978. Reprinted by permission of Intel Corporation.

Mnemonic: **IN and INW**

Description: IN (or INW) transfers a byte (or word) from an input port to the AL register (or AX register for INW). The port is specified either with an inline data byte, allowing fixed access to ports 0 through 255, or with a port number in the DX register, allowing variable access to 64K input ports.

Encoding:

Fixed Port:

1 1 1 0 0 1 0 w	port

if w = 0 then SRC = port, DEST = AL
else SRC = port+1:port, DEST = AX
Timing: 10 clocks

Variable Port:

1 1 1 0 1 1 0 w

if w = 0 then SRC = (DX), DEST = AL
else SRC = (DX)+1:(DX), DEST = AX
Timing: 8 clocks

Operation:

(DEST) <== (SRC)

Flags Affected:

None

Mnemonic: **OUT and OUTW**

Description: OUT (or OUTW) transfers a byte (or word) from the AL register (or AX register for OUTW) to an output port. The port is specified either with an inline data byte, allowing fixed access to ports 0 through 255, or with a port number in the DX register, allowing variable access to 64K output ports.

Encoding:

Fixed Port:

1 1 1 0 0 1 1 w	port

if w = 0 then SRC = AL, DEST = port
else SRC = AX, DEST = port+1:port
Timing: 10 clocks

Variable Port:

1 1 1 0 1 1 1 w

if w = 0 then SRC = AL, DEST = (DX)
else SRC = AX, DEST = (DX)+1:(DX)
Timing: 8 clocks

Operation:

(DEST) <== (SRC)

Flags Affected:

None

Mnemonic: **XLAT**

Description: XLAT performs a table lookup byte translation. The AL register is used as an index into a 256-byte table addressed by the BX register. The byte operand so addressed is transferred to AL.

Encoding:

1 1 0 1 0 1 1 1

Timing: 11 clocks

Operation:

(AL) <== ((BX) + (AL))

Flags Affected:

None

Three Address-Object transfer operations are provided:

Address Object Transfers
LEA Load Effective Address
LDS Load Pointer into DS
LES Load Pointer into ES

Mnemonic: **LEA**

Description: LEA (Load Effective Address) transfers the offset address of the source operand to the destination operand. The source operand must be a memory operand and the destination operand can be any 16-bit general, pointer, or index register.

Encoding:

1 0 0 0 1 1 0 1	mod reg r/m

if mod = 11 then undefined operation
Timing: 2+EA clocks

Operation:

(REG) <== EA

Flags Affected:

None

Mnemonic: **LDS**

Description: LDS (Load Pointer into DS) transfers a "pointer-object" (i.e., a 32-bit object containing an offset address and a segment address) from the source operand (which must be a memory operand) to a pair of destination registers. The segment address is transferred to the DS segment register. The offset address may be transferred to any 16-bit general, pointer, or index register.

Encoding:

1 1 0 0 0 1 0 1	mod reg r/m

if mod = 11 then undefined operation
Timing: 16+EA clocks

Operation:

(REG) <== (EA)
(DS) <== (EA + 2)

Flags Affected:

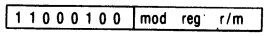
None

From *MCS-86 User's Manual.* Mnemonics © Intel, 1978. Reprinted by permission of Intel Corporation.

Mnemonic: **LES**

Description: LES (Load Pointer into ES) transfers a "pointer object" (i.e., a 32-bit object containing an offset address and a segment address) from the source operand (which must be a memory operand) to a pair of destination registers. The segment address is transferred to the ES segment register. The offset address may be transferred to a 16-bit general, pointer, or index register.

Encoding:



if mod = 11 then undefined operation
Timing: 16+EA clocks

Operation:

(REG) <== (EA)
(ES) <== (EA + 2)

Flags Affected:

None

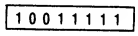
Four Flag Register transfer operations are provided:

LAHF	Load AH with Flags
SAHF	Store AH into Flags
PUSHF	Push Flags
POPF	Pop Flags

Mnemonic: **LAHF**

Description: LAHF (Load AH with Flags) transfers the flag registers SF, ZF, AF, PF, and CF (which, when 8080 code is translated into 8086 code, are the 8080 flags) into specific bits of the AH register. The bits indicated "X" are unspecified.

Encoding:



Timing: 4 clocks

Operation:

(AH) <== (SF):(ZF):X:(AF):X:(PF):X:(CF)

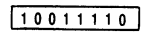
Flags Affected:

None

Mnemonic: **SAHF**

Description: SAHF transfers specific bits of the AH register to the flag registers SF, ZF, AF, PF, and CF. The bits of AH indicated by "X" in the operation below are ignored.

Encoding:



Timing: 4 clocks

Operation:

(SF):(ZF):X:(AF):X:(PF):X:(CF) <== (AH)

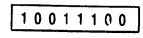
Flags Affected:

AF, CF, PF, SF, ZF

Mnemonic: **PUSHF**

Description: PUSHF decrements the SP register by 2 and transfers all of the flag registers into specific bits of the word operand (stack element) addressed by SP.

Encoding:



Timing: 10 clocks

Operation:

(SP) <== (SP) - 2
((SP)+1:(SP)) <== Flags

Flags Affected:

None

From *MCS-86 User's Manual.* Mnemonics © Intel, 1978. Reprinted by permission of Intel Corporation.

Mnemonic: **POPF**

Description: POPF (pop flags) transfers specific bits of the stack element addressed by the SP register to the flag registers and then increments SP by two.

Encoding:

1 0 0 1 1 1 0 1

Timing: 8 clocks

Operation:

Flags <== ((SP)+1:(SP))
(SP) <== (SP) + 2

Flags Affected:

All

Arithmetic

The 8086 provides the four basic mathematical operations in a number of different varieties. Both 8- and 16-bit operations and both signed and unsigned arithmetic are provided. Standard two's complement representation of signed values is used. The addition and subtraction operations serve as both signed and unsigned operations. In these cases the flag settings allow the distinction between signed and unsigned operations to be made (see Conditional Transfer). Correction operations are provided to allow arithmetic to be performed directly on unpacked decimal digits or on packed decimal representations.

Six flag bits are set or cleared by most arithmetic operations to reflect certain properties of the result of the operation. They generally follow these rules:

- CF is set if the operation resulted in a carry out of (from addition) or a borrow into (from subtraction) the high-order bit of the result; otherwise CF is cleared.
- AF is set if the operation resulted in a carry out of (from addition) or a borrow into (from subtraction) the low-order four bits of the result; otherwise AF is cleared.
- ZF is set if the result of the operation is zero; otherwise ZF is cleared.
- SF is set if the high-order bit of the result of the operation is set; otherwise SF is cleared.
- PF is set if the modulo 2 sum of the low-order eight bits of the result of the operation is 0 (even parity); otherwise PF is cleared (odd parity).
- OF is set if the operation resulted in a carry into the high-order bit of the result but not a carry out of the high-order bit, or vice versa; otherwise OF is cleared.

Five addition operations are provided:

ADD Add
ADC Add with Carry
INC Increment
AAA Unpacked BCD (ASCII) Adjust for Addition
DAA Decimal Adjust for Addition

Mnemonic: **ADD**

Description: ADD performs an addition of the two source operands and returns the result to one of the operands.

Encoding:

Memory or Register Operand with Register Operand:

0 0 0 0 0 0 d w	mod reg r/m

if d = 1 then LSRC = REG, RSRC = EA, DEST = REG
else LSRC = EA, RSRC = REG, DEST = EA
Timing (clocks): register to register 3
memory to register 9+EA
register to memory 16+EA

Immediate Operand to Memory or Register Operand:

1 0 0 0 0 0 s w	mod 0 0 0 r/m	data	data if s:w=01

LSRC = EA, RSRC = data, DEST = EA
Timing (clocks): immediate to memory 17+EA
immediate to register 4

Immediate Operand to Accumulator:

0 0 0 0 0 1 0 w	data	data if w=1

if w = 0 then LSRC = AL, RSRC = data, DEST = AL
else LSRC = AX, RSRC = data, DEST = AX
Timing: 4 clocks

Operation:

(DEST) <== (LSRC) + (RSRC)

Flags Affected:

AF, CF, OF, PF, SF, ZF

Mnemonic: **ADC**

Description: ADC (add with carry) performs an addition of the two source operands, adds one if the CF flag is set, and returns the result to one of the operands.

Encoding:

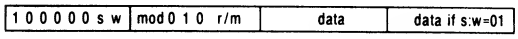
Memory or Register Operand with Register Operand:

0 0 0 1 0 0 d w	mod reg r/m

if d = 1 then LSRC = REG, RSRC = EA, DEST = REG
else LSRC = EA, RSRC = REG, DEST = EA
Timing (clocks): register to register 3
memory to register 9+EA
register to memory 16+EA

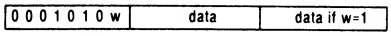
From *MCS-86 User's Manual*. Mnemonics © Intel, 1978. Reprinted by permission of Intel Corporation.

Immediate Operand to Memory or Register Operand:



LSRC = EA, RSRC = data, DEST = EA
Timing (clocks): immediate to memory 17+EA
immediate to register 4

Immediate Operand to Accumulator:



if w = 0 then LSRC = AL, RSRC = data, DEST = AL
else LSRC = AX, RSRC = data, DEST = AX
Timing: 4 clocks

Operation:

if (CF) = 1 then (DEST) <== (LSRC) + (RSRC) + 1
else (DEST) <== (LSRC) + (RSRC)

Flags Affected:

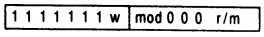
AF, CF, OF, PF, SF, ZF

Mnemonic: **INC**

Description: INC (increment) performs an addition of the source operand and one, and returns the result to the operand.

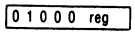
Encoding:

Memory or Register Operand:



DEST = EA
Timing (clocks): register 2
memory 15+EA

Register Operand:



DEST = REG
Timing: 2 clocks

Operation:

(DEST) <== (DEST) + 1

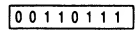
Flags Affected:

AF, OF, PF, SF, ZF

Mnemonic: **AAA**

Description: AAA (Unpacked BCD (ASCII) adjust for addition) performs a correction of the result in AL of adding two unpacked decimal operands, yielding an unpacked decimal sum.

Encoding:



Timing: 4 clocks

Operation:

if ((AL) & OFH) > 9 or (AF) = 1 then
(AL) <== (AL) + 6
(AH) <== (AH) + 1
(AF) <== 1
(CF) <== (AF)
(AL) <== (AL) & OFH

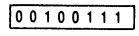
Flags Affected:

AF, CF.
OF, PF, SF, ZF undefined

Mnemonic: **DAA**

Description: DAA (decimal adjust for addition) performs a correction of the result in AL of adding two packed decimal operands, yielding a packed decimal sum.

Encoding:



Timing: 4 clocks

Operation:

if ((AL) & OFH) > 9 or (AF) = 1 then
(AL) <== (AL) + 6
(AF) <== 1
if (AL) > 9FH or (CF) = 1 then
(AL) <== (AL) + 6OH
(CF) <== 1

Flags Affected:

AF, CF, PF, SF, ZF
OF undefined

Seven subtractions operations are provided:

SUB	Subtract
SBB	Subtract with Borrow
DEC	Decrement
NEG	Negate
CMP	Compare
AAS	Unpacked BCD (ASCII) Adjust for Subtraction
DAS	Decimal Adjust for Subtraction

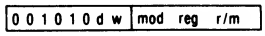
From *MCS-86 User's Manual.* Mnemonics © Intel, 1978. Reprinted by permission of Intel Corporation.

Mnemonic: **SUB**

Description: SUB performs a subtraction of the two source operands and returns the result to one of the operands.

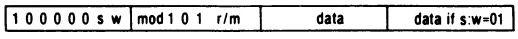
Encoding:

Memory or Register Operand and Register Operand:



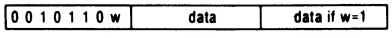
if d = 1 then LSRC = REG, RSRC = EA, DEST = REG
else LSRC = EA, RSRC = REG, DEST = EA
Timing (clocks): register from register 3
memory from register 9+EA
register from memory 16+EA

Immediate Operand from Memory or Register Operand:



LSRC = EA, RSRC = data, DEST = EA
Timing (clocks): immediate from register 4
immediate from memory 17+EA

Immediate Operand from Accumulator:



if w = 0 then LSRC = AL, RSRC = data, DEST = AL
else LSRC = AX, RSRC = data, DEST = AX
Timing (clocks): immediate from register 4

Operation:

(DEST) <== (LSRC) - (RSRC)

Flags Affected:

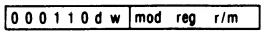
AF, CF, OF, PF, SF, ZF

Mnemonic: **SBB**

Description: SBB (subtract with borrow) performs a subtraction of the two source operands, subtracts one if the CF flag is set, and returns the result to one of the operands.

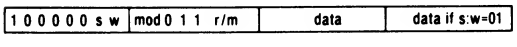
Encoding:

Memory or Register Operand and Register Operand:



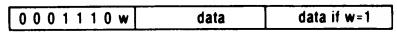
if d = 1 then LSRC = REG, RSRC = EA, DEST = REG
else LSRC = EA, RSRC = REG, DEST = EA
Timing (clocks): register from register 3
memory from register 9+EA
register from memory 16+EA

Immediate Operand from Memory or Register Operand:



LSRC = EA, RSRC = data, DEST = EA
Timing (clocks): immediate from register 4
immediate from memory 17+EA

Immediate Operand from Accumulator:



if w = 0 then LSRC = AL, RSRC = data, DEST = AL
else LSRC = AX, RSRC = data, DEST = AX
Timing: (clocks): immediate from register 4

Operation:

if (CF) = 1 then (DEST) <== (LSRC) - (RSRC) - 1
else (DEST) <== (LSRC) - (RSRC)

Flags Affected:

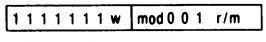
AF, CF, OF, PF, SF, ZF

Mnemonic: **DEC**

Description: DEC (decrement) performs a subtraction of one from the source operand and returns the result to the operand.

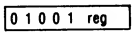
Encoding:

Memory or Register Operand:



DEST = EA
Timing (clocks): register 2
memory 15+EA

Register Operand:



DEST = REG
Timing: 2 clocks

Operation:

(DEST) <== (DEST) - 1

Flags Affected:

AF, OF, PF, SF, ZF

From *MCS-86 User's Manual.* Mnemonics © Intel, 1978. Reprinted by permission of Intel Corporation.

Mnemonic: **NEG**

Description: NEG (negate) performs a subtraction of the source operand from zero and returns the result to the operand.

Encoding:

1 1 1 1 0 1 1 w	mod 0 1 1 r/m

if w = 0 then SRC = FFH
else SRC = FFFFH
Timing (clocks): register 3
memory 16+EA

Operation:

(EA) <== SRC - (EA)
(EA) <== (EA) + 1 (affecting flags)

Flags Affected:

AF, CF, OF, PF, SF, ZF

Mnemonic: **CMP**

Description: CMP (compare) performs a subtraction of the two source operands causing the flags to be affected but does not return the result.

Encoding:

Memory or Register Operand with Register Operand:

0 0 1 1 1 0 d w	mod reg r/m

if d = 1 then LSRC = REG, RSRC = EA
else LSRC = EA, RSRC = REG
Timing (clocks): register with register 3
memory with register 9+EA
register with memory 16+EA

Immediate Operand with Memory or Register Operand:

1 0 0 0 0 0 s w	mod 1 1 1 r/m	data	data if s:w=01

LSRC = EA, RSRC = data
Timing (clock): immediate with register 4
immediate with memory 17+EA

Immediate Operand with Accumulator:

0 0 1 1 1 1 0 w	data	data if w=1

if w = 0 then LSRC = AL, RSRC = data
else LSRC = AX, RSRC = data
Timing (clocks): immediate with register 4

Operation:

(LSRC) - (RSRC)

Flags Affected:

AF, CF, OF, PF, SF, ZF

Mnemonic: **AAS**

Description: AAS (Unpacked BCD (ASCII) adjust for subtraction) performs a correction of the result in the AL register of subtracting two unpacked decimal operands, yielding an unpacked decimal difference.

Encoding:

0 0 1 1 1 1 1 1

Timing: 4 clocks

Operation:

if ((AL) & OFH) > 9 or (AF) = 1 then
(AL) <== (AL) - 6
(AH) <== (AH) - 1
(AF) <== 1
(CF) <== (AF)
(AL) <== (AL) & OFH

Flags Affected:

AF, CF.
OF, PF, SF, ZF undefined

Mnemonic: **DAS**

Description: DAS (decimal adjust for subtraction) performs a correction of the result in the AL register of subtracting two packed decimal operands, yielding a packed decimal difference.

Encoding:

0 0 1 0 1 1 1 1

Timing: 4 clocks

Operation:

if ((AL) & OFH) > 9 or (AF) = 1 then
(AL) <== (AL) - 6
(AF) <== 1
if (AL) > 9FH or (CF) = 1 then
(AL) <== (AL) - 6OH
(CF) <== 1

Flags Affected:

AF, CF, PF, SF, ZF.
OF undefined

Three multiplication operations are provided:

MUL	Multiply
IMUL	Integer Multiply
AAM	Unpacked BCD (ASCII) Adjust for Multiplication

From *MCS-86 User's Manual.* Mnemonics © Intel, 1978. Reprinted by permission of Intel Corporation.

Mnemonic: **MUL**

Description: MUL (multiply) performs an unsigned multiplication of the accumulator (AL or AX) and the source operand, returning a double-length result to the accumulator and its extension (AL and AH for 8-bit operation, or AX and DX for 16-bit operation). CF and OF are set if the top half of the result is nonzero.

Encoding:

1 1 1 1 0 1 1 w	mod 1 0 0 r/m

if w = 0 then LSRC = AL, RSRC = EA, DEST = AX, EXT = AH
else LSRC = AX, RSRC = EA, DEST = DX:AX, EXT = DX
Timing (clocks): 8-bit 71+EA
16-bit 124+EA

Operation:

(DES) <== (LSRC) * (RSRC), where * is unsigned multiply
if (EXT) = 0 then (CF) <== 0
else (CF) <== 1;
(OF) <== (CF)

Flags Affected:

CF, OF.
AF, PF, SF, ZF undefined

Mnemonic: **IMUL**

Description: IMUL (integer multiply) performs a signed multiplication of the accumulator (AL or AX) and the source operand, returning a double-length result to the accumulator and its extension (AL and AH for 8-bit operation, or AX and DX for 16-bit operation). CF and OF are set if the top half of the result is not the sign-extension of the low half of the result.

Encoding:

1 1 1 1 0 1 1 w	mod 1 0 1 r/m

if w = 0 then LSRC = AL, RSRC = EA, DEST = AX, EXT = AH, LOW = AL
else LSRC = AX, RSRC = EA, DEST = DX:AX, EXT = DX, LOW = AX
Timing (clocks): 8-bit 90+EA
16-bit 144+EA

Operation:

(DEST) <== (LSRC) * (RSRC) where * is signed multiply
if (EXT) = sign-extension of (LOW) then (CF) <== 0
else (CF) <== 1;
(OF) <== (CF)

Flags Affected:

CF, OF.
AF, PF, SF, ZF undefined

Mnemonic: **AAM**

Description: AAM (Unpacked BCD (ASCII) adjust for multiply) performs a correction of the result in AX of multiplying two unpacked decimal operands, yielding an unpacked decimal product.

Encoding:

1 1 0 1 0 1 0 0	0 0 0 0 1 0 1 0

Timing: 83 clocks

Operation:

(AH) <== (AL) / OAH
(AL) <== (AL) % OAH

Flags Affected:

PF, SF, ZF.
AF, CF, OF undefined

Three division operations are provided, as well as two sign-extension operations which support signed division.

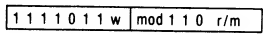
DIV Divide
IDIV Integer Divide
AAD Unpacked BCD (ASCII) Adjust for Division
CBW Convert Byte to Word
CWD Convert Word to Double Word

From *MCS-86 User's Manual.* Mnemonics © Intel, 1978. Reprinted by permission of Intel Corporation.

Mnemonic: **DIV**

Description: DIV (divide) performs an unsigned division of the double-length NUMR operand, contained in the accumulator and its extension (AL and AH for 8-bit operation, or AX and DX for 16-bit operation) by the DIVR operand, contained in the specified source operand. It returns the single-length quotient (QUO operand) to the accumulator (AL or AX), and returns the single-length remainder (the REM operand) to the accumulator extension (AH for 8-bit operation or DX for 16-bit operation). If the quotient is greater than MAX (as when division by zero is attempted) then QUO and REM are undefined, and a type 0 interrupt is generated. Flags are undefined in any DIV operation. Nonintegral quotients are truncated to integers.

Encoding:



if w = 0 then NUMR = AX, DIVR = EA, QUO = AL, REM = AH, MAX = FFH
else NUMR = DX:AX, DIVR = EA, QUO = AX, REM = DX, MAX = FFFFH
Timing: (clocks): 8-bit 90+EA
16-bit 155+EA

Operation:

```
(temp) <== (NUMR)
if (temp) / (DIVR) > MAX then the following, in sequence
 (QUO), (REM) undefined
 (SP) <== (SP) - 2
 ((SP)+1:(SP)) <== FLAGS
 (IF) <== 0
 (TF) <== 0
 (SP) <== (SP) - 2
 ((SP)+1:(SP)) <== (CS)
 (CS) <== (2) i.e., the contents of memory locations 2
   and 3
 (SP) <== (SP) - 2
 ((SP)+1:(SP)) <== (IP)
 (IP) <== (0) i.e., the contents of locations 0 and 1
else
 (QUO) <== (temp) / (DIVR), where / is unsigned
   division
 (REM) <== (temp) % (DIVR), where % is unsigned
   modulo
```

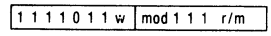
Flags Affected:

AF, CF, OF, PF, SF, ZF undefined

Mnemonic: **IDIV**

Description: IDIV (integer divide) performs a signed division of the double-length NUMR operand, contained in the accumulator and its extension (AL and AH for 8-bit operation, or AX and DX for 16-bit operation) by the DIVR operand, contained in the specified source operand. It returns the single-length quotient (QUO operand) to the accumulator (AL or AX), and returns the single-length remainder (the REM operand) to the accumulator extension (AH for 8-bit operation or DX for 16-bit operation). If the quotient is positive and greater than MAX or if the quotient is negative and less than (0 - MAX - 1), (as when division by zero is attempted) then QUO and REM are undefined, and a type 0 interrupt is generated. Flags are undefined in any divide operation. IDIV truncates nonintegral quotients and returns a remainder with the same sign as the numerator.

Encoding:



if w = 0 then NUMR = AX, DIVR = EA, QUO = AL, REM = AH, MAX = 7FH
else NUMR = DX:AX, DIVR = EA, QUO = AX, REM = DX, MAX = 7FFFH
Timing (clocks): 8-bit 112+EA
16-bit 177+EA

Operation:

```
(temp) <== (NUMR)
if (temp) / (DIVR) > 0 and (temp) / (DIVR) > MAX
or (temp) / (DIVR) < 0 and (temp) / (DIVR) < 0 - MAX - 1
  then
 (QUO), (REM) undefined
 (SP) <== (SP) - 2
 ((SP)+1:(SP)) <== FLAGS
 (IF) <== 0
 (TF) <== 0
 (SP) <== (SP) - 2
 ((SP)+1:(SP)) <== (CS)
 (CS) <== (2)
 (SP) <== (SP) - 2
 ((SP)+1:(SP)) <== (IP)
 (IP) <== (0)
else
 (QUO) <== (temp) / (DIVR), where / is signed division
 (REM) <== (temp) % (DIVR), where % is signed modulo
```

Flags Affected:

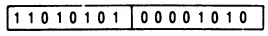
AF, CF, OF, PF, SF, ZF undefined

From *MCS-86 User's Manual.* Mnemonics © Intel, 1978. Reprinted by permission of Intel Corporation.

Mnemonic: **AAD**

Description: AAD (Unpacked BCD (ASCII) adjust for division) performs an adjustment of the dividend in AL before dividing two unpacked decimal operands, so that the result of the division will be an unpacked decimal quotient.

Encoding:



Timing: 60 clocks

Operation:

(AL) <== (AH) * OAH + (AL)
(AH) <== 0

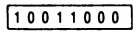
Flags Affected:

PF, SF, ZF.
AF, CF, OF undefined

Mnemonic: **CBW**

Description: CBW (convert byte to word) performs a sign extension of the AL register into the AH register.

Encoding:



Timing: 2 clocks

Operation:

if (AL) < 80H then (AH) <== 0 else (AH) <== FFH

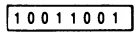
Flags Affected:

None

Mnemonic: **CWD**

Description: CWD (convert word to double word) performs a sign extension of the AX register into the DX register.

Encoding:



Timing: 5 clocks

Operation:

if (AX) < 8000H then (DX) <== 0
else (DX) <== FFFFH

Flags Affected:

None

Logic

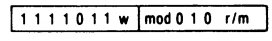
Nine single-operand logical instructions are provided:

NOT	Not
SHL	Shift Left
SAL	Shift Arithmetic Left
SHR	Shift Right
SAR	Shift Arithmetic Right
ROL	Rotate Left
ROR	Rotate Right
RCL	Rotate through Carry Left
RCR	Rotate through Carry Right

Mnemonic: **NOT**

Description: NOT forms the ones complement of (inverts) the source operand and returns the result to the operand. Flags are not affected.

Encoding:



if w = 0 then SRC = FFH
else SRC = FFFFH
Timing (clocks): register 3
memory 16+EA

Operation:

(EA) <== SRC - (EA)

Flags Affected:

None

From *MCS-86 User's Manual.* Mnemonics © Intel, 1978. Reprinted by permission of Intel Corporation.

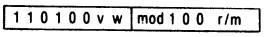
SHIFTS

Shift operations of four varieties are provided for memory and register operands, SHL (shift logical left), SHR (shift logical right), SAL (shift arithmetic left), and SAR (shift arithmetic right). Single bit shifts, and variable bit shifts with the shift count taken from the CL register are available. The CF flag becomes the last bit shifted out; OF is set if the final sign bit value of a single shift differs from the previous value of the sign bit; and PF, SF, and ZF are set to reflect the result value.

Mnemonic: **SHL and SAL**

Description: SHL (shift logical left) and SAL (shift arithmetic left) shift the source operand left by COUNT bits, shifting in low-order zero bits.

Encoding:



if v = 0 then COUNT = 1
else COUNT = (CL)
Timing (clocks): single-bit register 2
single-bit memory 15+EA
variable-bit register 8+4/bit
variable-bit memory 20+EA+4/bit

Operation:

```
(temp) <== COUNT
do while (temp) ≠ 0
 (CF) <== high-order bit of (EA)
 (EA) <== (EA) * 2
 (temp) <== (temp) - 1
if COUNT = 1 then
 if high-order bit of (EA) ≠ (CE) then (OF) <== 1
 else (OF) <== 0
else (OF) undefined
```

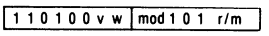
Flags Affected:

CF, OF, PF, SF, ZF.
AF undefined

Mnemonic: **SHR**

Description: SHR (shift logical right) shifts the source operand right by COUNT bits, shifting in high-order zero bits.

Encoding:



if v = 0 then COUNT = 1
else COUNT = (CL)
Timing (clocks): single-bit register 2
single-bit memory 15+EA
variable-bit register 8+4/bit
variable-bit memory 20+EA+4/bit

Operation:

```
(temp) <== COUNT
do while (temp) ≠ 0
 (CF) <== low-order bit of (EA)
 (EA) <== (EA) / 2, where / is equivalent to unsigned
   division
 (temp) <== (temp) - 1
if COUNT = 1 then
 if high-order bit of (EA) ≠ next-to-high-order bit of (EA)
   then (OF) <== 1
 else (OF) <== 0
else (OF) undefined
```

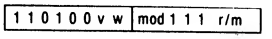
Flags Affected:

CF, OF, PF, SF, ZF.
AF undefined

Mnemonic: **SAR**

Description: SAR (shift arithmetic right) shifts the source operand right by COUNT bits, shifting in high-order bits equal to the original high-order bit of the operand (sign extension).

Encoding:



if v = 0 then COUNT = 1
else COUNT = (CL)
Timing (clocks): single-bit register 2
single-bit memory 15+EA
variable-bit register 8+4/bit
variable-bit memory 20+EA+4/bit

Operation:

```
(temp) <== COUNT
do while (temp) ≠ 0
 (CF) <== low-order bit of (EA)
 (EA) <== (EA) / 2, where / is equivalent to signed
   division, rounding down
 (temp) <== (temp) - 1
if COUNT = 1 then
 if high-order bit of (EA) ≠ next-to-high-order bit of (EA)
   then (OF) <== 1
else (OF) <== 0
else (OF) <== 0
```

Flags Affected:

CF, OF, PF, SF, ZF.
AF undefined

From *MCS-86 User's Manual.* Mnemonics © Intel, 1978. Reprinted by permission of Intel Corporation.

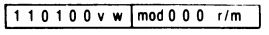
ROTATES

Rotate operations of four varieties are provided for memory and register operands, ROL (rotate left), ROR (rotate right), RCL (rotate through CF left), and RCR (rotate through CF right). Single bit rotates, and variable bit rotates with the rotate count taken from the CL register are available. The CF flag becomes the last bit rotated out; OF is set if the final sign bit value differs from the previous value of the sign bit.

Mnemonic: **ROL**

Description: ROL (rotate left) rotates the source operand left by COUNT bits.

Encoding:



if v = 0 then COUNT = 1
else COUNT = (CL)

Timing (clocks):		
	single-bit register	2
	single-bit memory	15+EA
	variable-bit register	8+4/bit
	variable-bit memory	20+EA+4/bit

Operation:

```
(temp) <== COUNT
do while (temp) ≠ 0
 (CF) <== high-order bit of (EA)
 (EA) <== (EA) * 2 + (CF)
 (temp) <== (temp) -1
if COUNT = 1 then
 if high-order bit of (EA) ≠ (CF) then (OF) <== 1
 else (OF) <== 0
else (OF) undefined
```

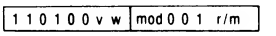
Flags Affected:

CF, OF

Mnemonic: **ROR**

Description: ROR (rotate right) rotates the source operand right by COUNT bits.

Encoding:



if v = 0 then COUNT = 1
else COUNT = (CL)

Timing (clocks):		
	single-bit register	2
	single-bit memory	15+EA
	variable-bit register	8+4/bit
	variable-bit memory	20+EA+4/bit

Operation:

```
(temp) <== COUNT
do while (temp) ≠ 0
 (CF) <== low-order bit of (EA)
 (EA) <== (EA) / 2
 high-order bit of (EA) <== (CF)
 (temp) <== (temp) -1
if COUNT = 1 then
 if high-order bit of (EA) ≠ next-to-high-order bit of (EA)
   then (OF) <== 1
 else (OF) <== 0
else (OF) undefined
```

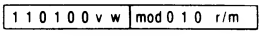
Flags Affected:

CF, OF

Mnemonic: **RCL**

Description: RCL (rotate through carry flag left) rotates the source operand left through the CF flag register by COUNT bits.

Encoding:



if v = 0 then COUNT = 1
else COUNT = (CL)

Timing (clocks):		
	single-bit register	2
	single-bit memory	15+EA
	variable-bit register	8+4/bit
	variable-bit memory	20+EA+4/bit

Operation:

```
(temp) <== COUNT
do while (temp) ≠ 0
 (tmpcf) <== (CF)
 (CF) <== high-order bit of (EA)
 (EA) <== (EA) * 2 + (tmpcf)
 (temp) <== (temp) - 1
if COUNT = 1 then
 if high-order bit of (EA) ≠ (CF) then (OF) <== 1
 else (OF) <== 0
else (OF) undefined
```

Flags Affected:

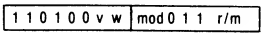
CF, OF

From *MCS-86 User's Manual*. Mnemonics © Intel, 1978. Reprinted by permission of Intel Corporation.

Mnemonic: **RCR**

Description: RCR (rotate through carry flag right) rotates the EA operand right through the CF flag register by COUNT bits.

Encoding:



if v = 0 then COUNT = 1
else COUNT = (CL)
Timing (clocks): single-bit register 2
single-bit memory 15+EA
variable-bit register 8+4/bit
variable-bit memory 20+EA+4/bit

Operation:

```
(temp) <== COUNT
do while (temp) ≠ 0
 (tmpcf) <== (CF)
 (CF) <== low-order bit of (EA)
 (EA) <== (EA) / 2
 high-order bit of (EA) <== (tmpcf)
 (temp) <== (temp) - 1
if COUNT = 1 then
 if high-order bit of (EA) ≠ next-to-high-order bit of (EA)
    then (OF) <== 1
 else (OF) <== 0
else (OF) undefined
```

Flags Affected:

CF, OF

Four two-operand logic operations are provided:

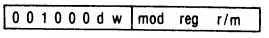
AND	And
TEST	Test
OR	Or
XOR	Exclusive Or

Mnemonic: **AND**

Description: AND performs the bitwise logical conjunction of the two source operands and returns the result to one of the operands.

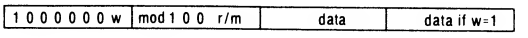
Encoding:

Memory or Register Operand with Register Operand:



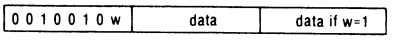
if d = 1 then LSRC = REG, RSRC = EA, DEST = REG
else LSRC = EA, RSRC = REG, DEST = EA
Timing (clocks): register to register 3
memory to register 9+EA
register to memory 16+EA

Immediate Operand to Memory or Register Operand:



LSRC = EA, RSRC = data, DEST = EA
Timing (clocks): immediate to register 4
immediate to memory 17+EA

Immediate Operand to Accumulator:



if w = 0 then LSRC = AL, RSRC = data, DEST = AL
else LSRC = AX, RSRC = data, DEST = AX
Timing (clocks): immediate to register 4

Operation:

```
(DEST) <== (LSRC) & (RSRC)
(CF) <== 0
(OF) <== 0
```

Flags Affected:

CF, OF, PF, SF, ZF.
AF undefined

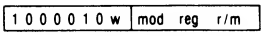
From *MCS-86 User's Manual.* Mnemonics © Intel, 1978. Reprinted by permission of Intel Corporation.

Mnemonic: **TEST**

Description: TEST performs the bitwise logical conjunction of the two source operands, causing the flags to be affected but does not return the result.

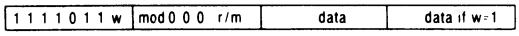
Encoding:

Memory or Register Operand with Register Operand:



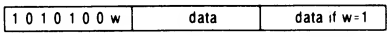
LSRC = REG, RSRC = EA
Timing (clocks): register with register 3
register with memory 9+EA

Immediate Operand with Memory or Register Operand:



LSRC = EA, RSRC = data
Timing (clocks): immediate with register 4
immediate with memory 10+EA

Immediate Operand with Accumulator:



if w = 0 then LSRC = AL, RSRC = data
else LSRC = AX, RSRC = data
Timing (clocks): immediate with register 4

Operation:

(LSRC) & (RSRC)
(CF) <== 0
(OF) <== 0

Flags Affected:

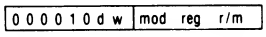
CF, OF, PF, SF, ZF.
AF undefined

Mnemonic: **OR**

Description: OR performs the bitwise logical inclusive disjunction of the two source operands and returns the result to one of the operands.

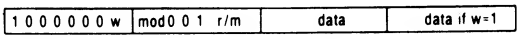
Encoding:

Memory or Register Operand with Register Operand:



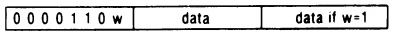
if d = 1 then LSRC = REG, RSRC = EA, DEST = REG
else LSRC = EA, RSRC = REG, DEST = EA
Timing (clocks): register to register 3
memory to register 9+EA
register to memory 16+EA

Immediate Operand to Memory or Register Operand:



LSRC = EA, RSRC = data, DEST = EA
Timing (clocks): immediate to register 4
immediate to memory 17+EA

Immediate Operand to Accumulator:



if w = 0 then LSRC = AL, RSRC = data, DEST = AL
else LSRC = AX, RSRC = data, DEST = AX
Timing (clocks): immediate to register 4

Operation:

(DEST) <== (LSRC) | (RSRC)
(CF) <== 0
(OF) <== 0

Flags Affected:

CF, OF, PF, SF, ZF.
AF undefined

From *MCS-86 User's Manual.* Mnemonics © Intel, 1978. Reprinted by permission of Intel Corporation.

Mnemonic: **XOR**

Description: XOR (exclusive Or) performs the bitwise logical exclusive disjunction of the two source operands and returns the result to one of the operands.

Encoding:

Memory or Register Operand with Register Operand:

0 0 1 1 0 0 d w	mod reg r/m

if d = 1 then LSRC = REG, RSRC = EA, DEST = REG
else LSRC = EA, RSRC = REG, DEST = EA
Timing (clocks): register to register 3
memory to register 9+EA
register to memory 16+EA

Immediate Operand to Memory or Register Operand:

1 0 0 0 0 0 0 w	mod 1 1 0 r/m	data	data if w=1

LSRC = EA, RSRC = data, DEST = EA
Timing (clocks): immediate to register 4
immediate to memory 17+EA

Immediate Operand to Accumulator:

0 0 1 1 0 1 0 w	data	data if w=1

if w = 0 then LSRC = AL, RSRC = data, DEST = AL
else LSRC = AX, RSRC = data, DEST = AX
Timing (clocks): immediate to register 4

Operation:

(DEST) <== (LSRC)⊕(RSRC)
(CF) <== 0
(OF) <== 0

Flags Affected:

CF, OF, PF, SF, ZF.
AF undefined

String Manipulation

The 8086 provides a group of one-byte instructions which perform various primitive operations for the manipulation of byte and word strings (sequences of bytes or words). These primitive operations can be performed repeatedly by preceding the instruction with a special prefix. The single-operation forms may be combined to form complex string operations with repetition provided by special interation operations.

Hardware Operation Control. All primitive string operations use the SI register to address the source operands, which are assumed to be in the current data segment. The DI register is used to address the destination operands, which are assumed to reside in the current extra segment. If the DF flag is cleared the operand pointers are incremented after each operation, once for byte operations and twice for word operations. If the DF flag is set the operand pointers are decremented after each operation. See Processor Control for setting and clearing DF.

Any of the primitive string operation instructions may be preceded with a one-byte prefix indicating that the operation is to be repeated until the operation count in CX is decremented to zero. The test for completion is made prior to each repetition of the operation. Thus, an initial operation count of zero will cause zero executions of the primitive operation.

The repeat prefix byte also designates a value to compare with the ZF flag. If the primitive operation is one which affects the ZF flag, and the ZF flag is unequal to the designated value after any execution of the primitive operation, the repetition is terminated. This permits the scan operation to serve as a scan-while or a scan-until.

During the execution of a repeated primitive operation the operand pointer registers (SI and DI) and the operation count register (CX) are updated after each repetition whereas the instruction pointer will retain the offset address of the repeat prefix byte (assuming it immediately precedes the string operation instruction). Thus, an interrupted repeated operation will be correctly resumed when control returns from the interrupting task.

A caution is in order at this point: It is possible to include two other special prefix bytes with a repeat-prefixed string instruction, one which overrides the default segment addressing for the SI operand (Section 2.3.1), and one which locks the bus to prohibit access by other bus masters. Execution of the repeated string operation will not resume properly following an interrupt if more than one prefix is present preceding the string primitive. Execution will resume one byte before the primitive (presumably where the repeat prefix resides), thus ignoring the additional prefixes.

Software Operation Control. The repeat prefix provides for rapid iteration in a hardware-repeated string operation. The iteration control operations (Section 4.4.7) provide this same control for implementing software loops to perform complex string operations. These iteration operations provide the same operation count update, operation completion test, and ZF flag tests that the repeat prefix provides.

By combining the primitive string operations and iteration control operations with other operations, it is possible to build sophisticated yet efficient string manipulation routines. One instruction that is particularly useful in this context is XLAT; it permits a byte fetched from one string to be translated before being stored in a second string, or before being operated upon in some other fashion. The translation is performed by using the value in the AL register to index into a table pointed at by the BX register. The translated value obtained from the table then replaces the value initially in the AL register.

As an example of the use of the primitive string operations and iteration control operations to implement a complex string operation, consider the following application: An input driver must translate a buffer of EBCDIC characters into ASCII, and transfer characters until one of several different EBCDIC control characters is encountered. The transferred ASCII string is to be terminated with an EOT character. To accomplish this, SI is initialized to point to the beginning of the EBCDIC buffer, DI is initialized to point to the beginning of the buffer to receive

From *MCS-86 User's Manual.* Mnemonics © Intel, 1978. Reprinted by permission of Intel Corporation.

the ASCII characters. BX is made to point to an EBCDIC to ASCII translation table, and CX is initialized to contain the length of the EBCDIC buffer (possibly empty). The translation table contains the ASCII equivalent for each EBCDIC character. perhaps with ASCII NULs for illegal characters. The EOT code is placed into those entries in the table corresponding to the desired EBCDIC stop characters. The 8086 instruction sequence to implement this example is the following:

```
        JCXZ    Empty   ;skip if input buffer empty
Next:   LODB    Ebcbuf  ;fetch next EBCDIC
                        character (from SI)
        XLAT    Table   ;translate it to ASCII (from
                        BX)
        CMP     AL,EOT  ;test for the EOT
        STOB    Ascbuf  ;translate ASCII character
                        (to DI)
        LOOPNE  Next    ;continue if not EOT
        •
        •
        •
Empty:
```

The body of this loop requires seven bytes of code.

Mnemonic: **REP**

Description: REP (repeat) causes the succeeding primitive string operation to be performed repeatedly while (CX) is not zero. In the case of CMPB, CMPW, SCAB, and SCAW, if after any repetition of the primitive operation the ZF flag differs from the "z" bit of the repeat prefix, the repetition is terminated. This prefix may be combined with the segment override and/or LOCK prefixes, although this has certain problems.

Encoding:

1 1 1 1 0 0 1 z

Timing: 6 clocks/loop

Operation:

```
do while (CX) ≠ 0
 service pending interrupt (if any)
 execute primitive string operation in succeeding byte
 (CX) <== (CX) - 1
 if primitive operation is CMPB, CMPW, SCAB, or
   SCAW and (ZF) ≠ z then exit from while loop
```

Flags Affected:

None

Five primitive string operations are provided, each of which has both a byte instruction and a word instruction, as follows:

MOVB	Move Byte
MOVW	Move Word
CMPB	Compare byte
CMPW	Compare Word
SCAB	Scan Byte
SCAW	Scan Word
LODB	Load byte
LODW	Load Word
STOB	Store Byte
STOW	Store Word

Mnemonic: **MOVB and MOVW**

Description: MOVB (or MOVW) transfers a byte (or word) operand from the source operand addressed by SI to the destination operand addressed by DI, and adjusts the SI and DI registers by DELTA. As a repeated operation this provides for moving a string from one location in memory to another.

Encoding:

1 0 1 0 0 1 0 w

if w = 0 then SRC = (SI), DEST = (DI), DELTA = 1
else SRC = (SI)+1:(SI), DEST = (DI)+1:(DI), DELTA = 2
Timing: 17 clocks

Operation:

```
(DEST) <== (SRC)
if (DF) = 0 then
 (SI) <== (SI) + DELTA
 (DI) <== (DI) + DELTA
else
 (SI) <== (SI) - DELTA
 (DI) <== (DI) - DELTA
```

Flags Affected:

None

From *MCS-86 User's Manual.* Mnemonics © Intel, 1978. Reprinted by permission of Intel Corporation.

Mnemonic: **CMPB and CMPW**

Description: CMPB (or CMPW) subtracts the destination byte (or word) operand addressed by DI from the source operand addressed by SI and affects the flags but does not return the result. As a repeated operation this provides for comparing two strings. With the appropriate repeat prefix it is possible to determine after which string element the two strings become unequal, thereby establishing an ordering between the strings.

Encoding:

1 0 1 0 0 1 1 w

if w = 0 then LSRC = (SI), RSRC = (DI), DELTA = 1
else LSRC = (SI)+1:(SI), RSRC = (DI)+1:(DI), DELTA = 2
Timing: 22 clocks

Operation:

(LSRC) - (RSRC)
if (DF) = 0 then
(SI) <== (SI) + DELTA
(DI) <== (DI) + DELTA
else
(SI) <== (SI) - DELTA
(DI) <== (DI) - DELTA

Flags Affected:

AF, CF, OF, PF, SF, ZF

Mnemonic: **SCAB and SCAW**

Description: SCAB (or SCAW) subtracts the destination byte (or word) operand addressed by DI from AL (or AX) and affects the flags but does not return the result. As a repeated operation this provides for scanning for the occurrence of, or departure from, a given value in a string.

Encoding:

1 0 1 0 1 1 1 w

if w = 0 then LSRC = AL, RSRC = (DI), DELTA = 1
else LSRC = AX, RSRC = (DI)+1:(DI), DELTA = 2
Timing: 15 clocks

Operation:

(LSRC) - (RSRC)
if (DF) = 0 then (DI) <== (DI) + DELTA
else (DI) <== (DI) - DELTA

Flags Affected:

AF, CF, OF, PF, SF, ZF

Mnemonic: **LODB and LODW**

Description: LODB (or LODW) transfers a byte (or word) operand from the source operand addressed by SI to accumulator AL (or AX) and adjusts the SI register by DELTA. This operation ordinarily would not be repeated.

Encoding:

1 0 1 0 1 1 0 w

if w = 0 then SRC = (SI), DEST = AL, DELTA = 1
else SRC = (SI)+1:(SI), DEST = AX, DELTA = 2
Timing: 12 clocks

Operation:

(DEST) <== (SRC)
if (DF) = 0 then (SI) <== (SI) + DELTA
else (SI) <== (SI) - DELTA

Flags Affected:

None

Mnemonic: **STOB and STOW**

Description: STOB (or STOW) transfers a byte (or word) operand from AL (or AX) to the destination operand addressed by DI and adjusts the DI register by DELTA. As a repeated operation this provides for filling a string with a given value.

Encoding:

1 0 1 0 1 0 1 w

if w = 0 then SRC = AL, DEST = (DI), DELTA = 1
else SRC = AX, DEST = (DI)+1:(DI), DELTA = 2
Timing: 10 clocks

Operation:

(DEST) <== (SRC)
if (DF) = 0 then (DI) <== (DI) + DELTA
else (DI) <== (DI) - DELTA

Flags Affected:

None

From *MCS-86 User's Manual.* Mnemonics © Intel, 1978. Reprinted by permission of Intel Corporation.

4.4.7 Control Transfer

Four classes of control transfer operations may be distinguished: calls, jumps, and returns; conditional transfers; iteration control; and interrupts.

All control transfer operations cause, perhaps upon a certain condition, the program execution to continue at some new location in memory, possibly in a new code segment.

NOTE: Queue reinitialization is not included in the timing information for transfer operations. To account for queue loading, add 4 clocks to timing numbers.

Calls, Jumps, and Returns. Two basic varieties of calls, jumps, and returns are provided — those which transfer control within the current code segment, and those which transfer control to an arbitrary code segment, which then becomes the current code segment. Both direct and indirect transfers are supported: indirect transfers make use of the standard addressing modes described in Section 4.4. Intra-segment direct calls and jumps specify a self-relative direct displacement, thus allowing position independent code. A shortened jump instruction is available for transfers within ±128 bytes from the instruction, using less code.

The three transfer operations are:

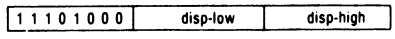
CALL	Call
JMP	Jump
RET	Return

Mnemonic: **CALL**

Description: CALL pushes the offset address of the next instruction onto the stack (in the case of an inter-segment transfer the CS segment register is pushed first) and then transfers control to the target operand.

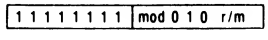
Encoding:

Intra-segment Direct:



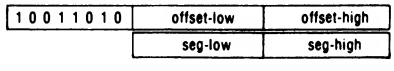
DEST = (EA)
Timing: 13+EA clocks

Intra-Segment Indirect:



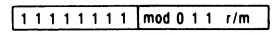
DEST = (IP) + disp
Timing: 11 clocks

Inter-Segment Direct:



DEST = offset, SEG = seg
Timing: 20 clocks

Inter-Segment Indirect:



DEST = (EA), SEG = (EA + 2)
Timing: 29+EA clocks

Operation:

```
if Inter-Segment then
  (SP) <== (SP) - 2
  ((SP)+1:(SP)) <== (CS)
  (CS) <== SEG
(SP) <== (SP) - 2
((SP)+1:(SP)) <== (IP)
(IP) <== DEST
```

Flags Affected:

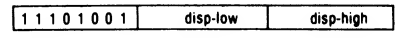
None

Mnemonic: **JMP**

Description: JMP transfers control to the target operand.

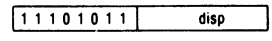
Encoding:

Intra-Segment Direct:



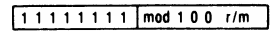
DEST = (IP) + disp
Timing: 7 clocks

Intra-Segment Direct Short:



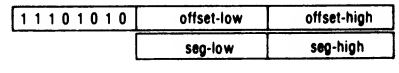
DEST = (IP) + disp sign extended to 16-bits
Timing: Timing: 2 clocks

Intra-Segment Indirect:



DEST = (EA)
Timing: 7+EA clocks

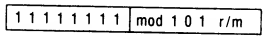
Inter-Segment Direct:



DEST = offset, SEG = seg
Timing: 7 clocks

From *MCS-86 User's Manual.* Mnemonics © Intel, 1978. Reprinted by permission of Intel Corporation.

Inter-Segment Indirect:



DEST = (EA), SEG = (EA + 2)
Timing: 16+EA clocks

Operation:

if Inter-Segment then (CS) <== SEG
(IP) <== DEST

Flags Affected:

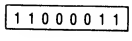
None

Mnemonic: **RET**

Description: RET transfers control to the return address pushed by a previous CALL operation and optionally adds an immediate constant to the SP register so as to discard stack parameters.

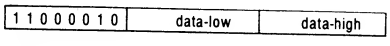
Encoding:

Intra-Segment:



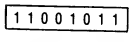
Timing: 8 clocks

Intra-Segment and Add Immediate to Stack Pointer:



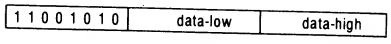
Timing: 12 clocks

Inter-Segment:



Timing: 18 clocks

Inter-Segment and Add Immediate to Stack Pointer:



Timing: 17 clocks

Operation:

(IP) <== ((SP)=1:(SP))
(SP) <== (SP) + 2
if Inter-Segment then
(CS) <== ((SP)+1:(SP))
(SP) <== (SP) + 2
if Add Immediate to Stack Pointer then (SP) <== (SP) + data

Flags Affected:

None

Conditional Transfers. The conditional transfers of control perform a jump contingent upon various Boolean functions of the flag registers. The destination must be within ±128 bytes distance from the instruction. Table 4-2 shows the available instructions, the conditions associated with them, and their interpretation.

TABLE 4-2.
8086 CONDITIONAL TRANSFER OPERATIONS

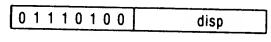
Instruction	Condition	Interpretation
JE or JZ	ZF = 1	"equal" or "zero"
JL or JNGE	(SF xor OF) = 1	"less" or "not greater or equal"
JLE or JNG	((SP xor OF) or ZF) = 1	"less or equal" or "not greater"
JB or JNAE	CF = 1	"below" or "not above or equal"
JBE or JNA	(CF or ZF) = 1	"below or equal" or "not above"
JP or JPE	PF = 1	"parity" or "parity even"
JO	OF = 1	"overflow"
JS	SF= 1	"sign"
JNE or JNZ	ZF = 0	"not equal" or "not zero"
JNL or JGE	(SF xor OF) = 0	"not less" or "greater or equal"
JNLE or JG	((SF xor OF) or ZF) = 0	"not less or equal" or "greater"
JNB or JAE	CF = 0	"not below" or "above or equal"
JNBE or JA	(CF or ZF) = 0	"not below or equal" or "above"
JNP or JPO	PF = 0	"not parity" or "parity odd"
JNO	OF = 0	"not overflow"
JNS	SF = 0	"not sign"

*"Above" and "below" refer to the relation between two unsigned values, while "greater" and "less" refer to the relation between two signed values.

Mnemonic: **JE and JZ**

Description: JE (or JZ) transfers control to the target operand on equal (or zero).

Encoding:



Timing (clocks): Jump is taken 8
Jump is not taken 4

Operation:

if (ZF) = 1 then
(IP) <== (IP) + disp (sign-extended to 16-bits)

Flags Affected:

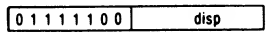
None

From *MCS-86 User's Manual*. Mnemonics © Intel, 1978. Reprinted by permission of Intel Corporation.

Mnemonic: **JL and JNGE**

Description: JL (or JNGE) transfers control to the target operand on less (or not greater or equal).

Encoding:



Timing (clocks): Jump is taken 8
Jump is not taken 4

Operation:

if (SF)$\overset{\|}{\|}$(OF) = 1 then
(IP) <== (IP) + disp (sign-extended to 16-bits)

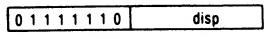
Flags Affected:

None

Mnemonic: **JLE and JNG**

Description: JLE (or JNG) transfers control to the target operand on less or equal (or not greater).

Encoding:



Timing (clocks): Jump is taken 8
Jump is not taken 4

Operation:

if [(SF)$\overset{\|}{\|}$(OF)] 1(ZF) = 1 then
(IP) <== (IP) + disp (sign-extended to 16-bits)

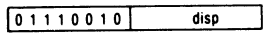
Flags Affected:

None

Mnemonic: **JB and JNAE**

Description: JB (or JNAE) transfers control to the target operand on below (or not above or equal).

Encoding:



Timing (clocks): Jump is taken 8
Jump is not taken 4

Operation:

if (CF) = 1 then
(IP) <== (IP) + disp (sign-extended to 16-bits)

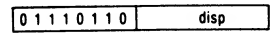
Flags Affected:

None

Mnemonic: **JBE and JNA**

Description: JBE (or JNA) transfers control to the target operand on below or equal (or not above).

Encoding:



Timing (clocks): Jump is taken 8
Jump is not taken 4

Operation:

if (CF)$\overset{|}{|}$(ZF) = 1 then
(IP) <== (IP) + disp (sign-extended to 16-bits)

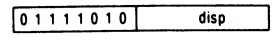
Flags Affected:

None

Mnemonic: **JP and JPE**

Description: JP (or JPE) transfers control to the target operand on parity (or parity even).

Encoding:



Timing (clocks): Jump is taken 8
Jump is not taken 4

Operation:

if (PF) = 1 then
(IP) <== (IP) + disp (sign-extended to 16-bits)

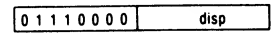
Flags Affected:

None

Mnemonic: **JO**

Description: JO transfers control to the target operand on overflow.

Encoding:



Timing (clocks): Jump is taken 8
Jump is not taken 4

Operation:

if (OF) = 1 then
(IP) <== (IP) + disp (sign-extended to 16-bits)

Flags Affected:

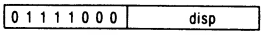
None

From *MCS-86 User's Manual.* Mnemonics © Intel, 1978. Reprinted by permission of Intel Corporation.

Mnemonic: **JS**

Description: JS transfers control to the target operand on sign.

Encoding:



Timing (clocks): Jump is taken 8
Jump is not taken 4

Operation:

if (SF) = 1 then
(IP) <== (IP) + disp (sign-extended to 16-bits)

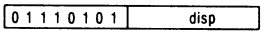
Flags Affected:

None

Mnemonic: **JNE and JNZ**

Description: JNE (or JNZ) transfers control to the target operand on not equal (or not zero).

Encoding:



Timing (clocks): Jump is taken 8
Jump is not taken 4

Operation:

if (ZF) = 0 then
(IP) <== (IP) + disp (sign-extended to 16-bits)

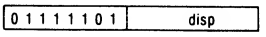
Flags Affected:

None

Mnemonic: **JNL and JGE**

Description: JNL (or JGE) transfers control to the target operand on not less (or greater or equal).

Encoding:



Timing (clocks): Jump is taken 8
Jump is not taken 4

Operation:

if (SF) ⊕ (OF) = 0 then
(IP) <== (IP) + disp (sign-extended to 16-bits)

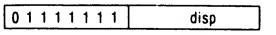
Flags Affected:

None

Mnemonic: **JNLE and JG**

Description: JNLE (or JG) transfers control to the target operand on not less or equal (or greater).

Encoding:



Timing (clocks): Jump is taken 8
Jump is not taken 4

Operation:

if ((SP ⊕ (OF)) ∨ (ZF) = 0 then
(IP) <== (IP) + disp (sign-extended to 16-bits)

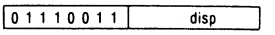
Flags Affected:

None

Mnemonic: **JNB and JAE**

Description: JNB (or JAE) transfers control to the target operand on not below (or above or equal).

Encoding:



Timing (clocks): Jump is taken 8
Jump is not taken 4

Operation:

if (CF) = 0 then
(IP) <== (IP) + disp (sign-extended to 16-bits)

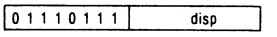
Flags Affected:

None

Mnemonic: **JNBE and JA**

Description: JNBE (or JA) transfers control to the target operand on not below or equal (or above).

Encoding:



Timing (clocks): Jump is taken 8
Jump is not taken 4

Operation:

if (CF) ∨ (ZF) = 0 then
(IP) <== (IP) + disp (sign-extended to 16-bits)

Flags Affected:

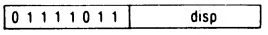
None

From *MCS-86 User's Manual.* Mnemonics © Intel, 1978. Reprinted by permission of Intel Corporation.

Mnemonic: **JNP and JPO**

Description: JNP (or JPO) transfers control to the target operand on not parity (or parity odd).

Encoding:



Timing (clocks): Jump is taken 8
Jump is not taken 4

Operation:

if (PF) = 0 then
(IP) <== (IP) + disp (sign-extended to 16-bits)

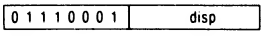
Flags Affected:

None

Mnemonic: **JNO**

Description: JNO transfers control to the target operand on not overflow.

Encoding:



Timing (clocks): Jump is taken 8
Jump is not taken 4

Operation:

if (OF) = 0 then
(IP) <== (IP) + disp (sign-extended to 16-bits)

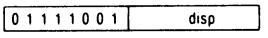
Flags Affected:

None

Mnemonic: **JNS**

Description: JNS transfers control to the target operand on not sign.

Encoding:



Timing (clocks): Jump is taken 8
Jump is not taken 4

Operation:

if (SF) = 0 then
(IP) <== (IP) + disp (sign-extended to 16-bits)

Flags Affected:

None

Iteration Control. The iteration control transfer operations perform leading- and trailing-decision loop control. The destination of iteration control transfers must be within ±128 bytes distance from the instruction. These operations are particularly useful in conjunction with the string manipulation operations. (See Section 4.4.6)

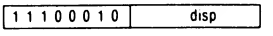
Four iteration control transfer operations are provided:

LOOP	Loop
LOOPZ	Loop While Zero
(LOOPE	Loop While Equal)
LOOPNZ	Loop While Not Zero
(LOOPNE	Loop While not Equal)
JCXZ	Jump on CX Zero

Mnemonic: **LOOP**

Description: LOOP decrements the CX (count) register by 1 and transfers control to the target operand if CX is not zero.

Encoding:



Timing (clocks): Jump is taken 9
Jump is not taken 5

Operation:

(CX) <== (CX) - 1
if (CX) ≠ 0 then
(IP) <== (IP) + disp (sign-extended to 16-bits)

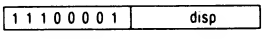
Flags Affected:

None

Mnemonic: **LOOPZ and LOOPE**

Description: LOOPZ, also called LOOPE (loop while zero or loop while equal) decrements the CX register by one and transfers if CX is not zero and if the ZF flag is set.

Encoding:



Timing (clocks): Jump is taken 11
Jump is not taken 5

Operation:

(CX) <== (CX) - 1
if (ZF) = 1 and (CX) ≠ 0 then
(IP) <== (IP) + disp (sign-extended to 16-bits)

Flags Affected:

None

From *MCS-86 User's Manual*. Mnemonics © Intel, 1978. Reprinted by permission of Intel Corporation.

Mnemonic: **LOOPNZ and LOOPNE**

Description: LOOPNZ, also called LOOPNE, (loop while not zero or loop while not equal) decrements the CX register by one and transfers if CX is not zero and the ZF flag is cleared.

Encoding:

1 1 1 0 0 0 0 0	disp

Timing (clocks): Jump is taken 11
Jump is not taken 5

Operation:

(CX) <== (CX) - 1
if (ZF) = 0 and (CX) ≠ 0 then
(IP) <== (IP) + disp (sign-extended to 16-bits)

Flags Affected:

None

Mnemonic: **JCXZ**

Description: JCXZ (jump on CX zero) transfers control to the target operand if the CX register is zero.

Encoding:

1 1 1 0 0 0 1 1	disp

Timing (clocks): Jump is taken 9
Jump is not taken 5

Operation:

if (CX) = 0 then
(IP) <== (IP) + disp (sign-extended to 16-bits)

Flags Affected:

None

Interrupts. Program execution control may be transferred by means of operations similar in effect to that of external interrupts (see Section 2.4.2). All interrupts perform a transfer by pushing the flag registers onto the stack (as in PUSHF), and then performing an indirect call (of the inter-segment variety) through an element of an interrupt transfer vector located at absolute locations 0 through 3FFH. This vector contains a four-byte element for each of up to 256 different interrupt types.

Three interrupt transfer operations are provided:

INT Interrupt
INTO Interrupt on Overflow
IRET Interrupt Return

Mnemonic: **INT**

Description: INT pushes the flag registers (as in PUSHF), clears the TF and IF flags, and transfers control with an indirect call through any one of the 256 vector elements. The one-byte form of this instruction generates a type 3 interrupt.

Encoding:

1 1 0 0 1 1 0 v	type if v=1

if v = 0 then TYPE = 3
else TYPE = type
Timing: 52 clocks

Operation:

(SP) <== (SP) - 2
((SP)+1:(SP)) <== FLAGS
(IF) <== 0
(TF) <== 0
(SP) <== (SP) - 2
((SP)+1:(SP)) <== (CS)
(CS) <== (TYPE * 4 + 2)
(SP) <==)SP) - 2
((SP)+1:(SP)) <== (IP)
(IP) <== (TYPE * 4)

Flags Affected:

IF, TF

Mnemonic: **INTO**

Description: INTO pushes the flag registers (as in PUSHF), clears the TF and IF flags, and transfers control with an indirect call through vector element 4 (location 1OH) if the OF flag is set (trap on overflow). If the OF flag is clear, no operation takes place.

Encoding:

1 1 0 0 1 1 1 0

Timing: 52 clocks

Operation:

if (OF) = 1 then
(SP) <== (SP) - 2
((SP)+1:(SP)) <== FLAGS
(IF) <== 0
(TF) <== 0
(SP) <== (SP) - 2
((SP)+1:(SP)) <== (CS)
(CS) <== (12H)
(SP) <== (SP) - 2
((SP)+1:(SP)) <== (IP)
(IP) <== (10H)

Flags Affected:

None

From *MCS-86 User's Manual.* Mnemonics © Intel, 1978. Reprinted by permission of Intel Corporation.

Mnemonic: **IRET**

Description: IRET transfers control to the return address saved by a previous interrupt operation and restores the saved flag registers (as in POPF).

Encoding:

1 1 0 0 1 1 1 1

Timing: 24 clocks

Operation:

(IP) <== ((SP)+1:(SP))
(SP) <== (SP) + 2
(CS) <== ((SP)+1:(SP))
(SP) <== (SP) + 2
FLAGS <== ((SP)+1:(SP))
(SP) <== (SP) + 2

Flags Affected:

All

4.4.8 Processor Control

The 8086 provides various instructions and mechanisms for controlling the operation of the processor and its interaction with its environment.

Flag Operations: Seven operations are provided which operate directly on individual flag registers:

CLC	Clear Carry
CMC	Complement Carry
STC	Set Carry
CLD	Clear Direction
STD	Set Direction
CLI	Clear Interrupt
STI	Set Interrupt

Mnemonic: **CLC**

Description: CLC clears the CF flag.

Encoding:

1 1 1 1 1 0 0 0

Timing: 2 clocks

Operation:

(CF) <== 0

Flags Affected:

CF

Mnemonic: **CMC**

Description: CMC complements the CF flag.

Encoding:

1 1 1 1 0 1 0 1

Timing: 2 clocks

Operation:

if (CF) = 0 then (CF) <== 1 else (CF) <== 0

Flags Affected:

CF

Mnemonic: **STC**

Description: STC sets the CF flag.

Encoding:

1 1 1 1 1 0 0 1

Timing: 2 clocks

Operation:

(CF) <== 1

Flags Affected:

CF

Mnemonic: **CLD**

Description: CLD clears the DF flag, causing the string operations to auto-increment the operand pointers.

Encoding:

1 1 1 1 1 1 0 0

Timing: 2 clocks

Operation:

(DF) <== 0

Flags Affected:

DF

Mnemonic: **STD**

Description: STD sets the DF flag, causing the string operations to auto-decrement the operand pointers.

Encoding:

1 1 1 1 1 1 0 1

Timing: 2 clocks

Operation:

(DF) <== 1

Flags Affected:

DF

From *MCS-86 User's Manual.* Mnemonics © Intel, 1978. Reprinted by permission of Intel Corporation.

Mnemonic: **CLI**

Description: CLI clears the IF flag, disabling maskable external interrupts, which appear on the INTR line of the 8086. (Nonmaskable interrupts, which appear on the NMI line, are not disabled).

Encoding:

1 1 1 1 1 0 1 0

Timing: 2 clocks

Operation:

(IF) <== 0

Flags Affected:

IF

Mnemonic: **STI**

Description: STI sets the IF flag, enabling maskable external interrupts after the execution of the next instruction.

Encoding:

1 1 1 1 1 0 1 1

Timing: 2 clocks

Operation:

(IF) <== 1

Flags Affected:

IF

Processor Halt

Mnemonic: **HLT**

Description: The HLT instruction causes the 8086 processor to enter its halt state. The halt state is cleared by an enabled external interrupt or reset.

Encoding:

1 1 1 1 0 1 0 0

Timing: 2 clocks

Operation:

None

Flags Affected:

None

Processor Wait

Mnemonic: **WAIT**

Description: The WAIT instruction causes the processor to enter a wait state if the signal on its $\overline{TEST}$ pin is not asserted. The wait state may be interrupted by an enabled external interrupt. When this occurs the saved code location is that of the WAIT instruction, so that upon return from the interrupting task the wait state is reentered. The wait state is cleared and execution resumed when the $\overline{TEST}$ signal is asserted. Execution resumes without allowing external interrupts until after the execution of the next instruction. This instruction allows the processor to synchonize itself with external hardware.

Encoding:

1 0 0 1 1 0 1 1

Timing: 3 clocks

Operation:

None

Flags Affected:

None

Processor Escape

Mnemonic: **ESC**

Description: The ESC instruction provides a mechanism by which other processors may receive their instructions from the 8086 instruction stream and make use of the 8086 addressing modes. The 8086 processor does no operation for the ESC instruction other than to access a memory operand and place it on the bus.

Encoding:

1 1 0 1 1 x	mod x r/m

Timing: 7+EA clocks

Operation:

if mod ≠ 11 then data bus <== (EA)

Flags Affected:

None

From *MCS-86 User's Manual.* Mnemonics © Intel, 1978. Reprinted by permission of Intel Corporation.

Bus Lock

Mnemonic: **LOCK**

Description: A special one-byte lock prefix may precede any instruction. It causes the processor to assert its bus-lock signal for the duration of the operation caused by that instruction. In multiple processor systems with shared resources it is necessary to provide mechanisms to enforce controlled access to those resources. Such mechanisms, while generally provided through software operating systems, require hardware assistance. A sufficient mechanism for accomplishing this is a locked exchange (also known as test-and-set-lock).

It is assumed that external hardware, upon receipt of that signal, will prohibit bus access for other bus masters during the period of its assertion.

The instruction most useful in this context is an exchange register with memory. A simple software lock may be implemented with the following code sequence:

```
Check:  MOV   AL1       ;set AL to 1 (implies locked)
  LOCK  XCHG  Sema,AL   ;test and set lock
        TEST  AL,AL     ;set flags based on AL
        JNZ   Check     ;retry if lock already set
        •
        •
        MOV   Sema,0    ;clear the lock when done
```

The LOCK prefix may be combined with the segment override and/or REP prefixes, although the latter has certain problems. (See Section 4.4.6.)

Encoding:

1 1 1 1 0 0 0 0

Timing: 2 clocks

Operation:

None

Flags Affected:

None

Single Step. When the TF flag register is set, the processor generates a type 1 interrupt after the execution of each instruction. During interrupt transfer sequences caused by any type of interrupt, the TF flag is cleared after the push-flags step of the interrupt sequence. No instructions are provided for setting or clearing TF directly. Rather, the flag register file image saved on the stack by a previous interrupt operation must be modified, so that the subsequent interrupt return operation (IRET) restores TF set. This allows a diagnostic task to single-step through a task under test, while still executing normally itself.

If the single-stepped instruction itself clears the TF flag, the type 1 interrupt will still occur upon completion of the single-stepped instruction. If the single-stepped instruction generates an interrupt or if an enabled external interrupt occurs prior to the completion of the single-stepped instruction, the type 1 interrupt sequence will occur after the interrupt sequence of the generated or external interrupt, but before the first instruction of that interrupt service routine is executed.

From *MCS-86 User's Manual*. Mnemonics © Intel, 1978. Reprinted by permission of Intel Corporation.

4

intel 8086 program development

The next step of microcomputer system design is that of program development. In this chapter we consider the development of programs for the Intel 8086 using assembly language. We examine the following topics:

- Assembly language programming.
- Effective address computation.
- Detailed instruction set.

Assembly Language Programming

An assembly language program for the Intel 8086 consists of a sequence of instructions using the mnemonics and instruction set presented in the preceding chapter. The basic format of an assembly language instruction is:

LABEL OPERATOR OPERAND

The leftmost field, LABEL, is merely a means by which the programmer may refer to that line in symbolic terms. The label field is some name that the programmer arbitrarily defines and is not actually used by the computer. The label field is optional.

The operator field contains the mnemonic for the instruction to be executed. The optional operand field contains one or two bytes of data or other information to be used by the instruction.

Finally, it is customary to write commentary or remarks following the operand field so that the purpose of the instruction set is more readily understood.

The assembly language program is translated by an assembler into machine language for the system. The assembly language program, or "source program," written in symbolic form by the programmer, is translated into 1's and 0's by the assembler, to form the "object program" that is actually run on the system.

To represent binary numbers without writing out 1 and 0 bit patterns, we use the hexadecimal number system. The hexadecimal numbers 0 through "F" correspond to binary numbers 0000 through 1111 as shown in Table 4.1. Using a hexadecimal representation simpli-

TABLE 4.1. Binary to Hexadecimal Representation

Binary	*Hexadecimal*
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	A
1011	B
1100	C
1101	D
1110	E
1111	F

fies the reference to specific memory locations and permits the mnemonic instruction to be written in a compact, machine-related form. Each 8-bit byte is broken up into two 4-bit segments:

xxxx xxxx

Each of these 4-bit segments can be written as a hexadecimal character.

For example, the 8086 instruction MOVE segment register to register/memory has as its first byte:

10001100

To translate this 8-bit binary number into a hexadecimal representation, we break it up into two 4-bit fields:

1000 1100

Using Table 4.1, we then translate each of these fields into the hexadecimal representation:

8 C

Thus 8C becomes the hexadecimal representation for the first byte of the particular MOVE instruction considered.

Since a knowledge of the hexadecimal representations is important for system development, the remainder of the present chapter is concerned with determining the hexadecimal representation of the mnemonic instruction.

Although the instruction set of the 8086 is listed in detail in Chapter 3, it would be convenient to summarize it here for use in this chapter. The 8086 instruction set summary is presented in Table 4.2.

Effective Address Computation

When an instruction makes reference to a memory location, the effective address must be computed according to the mod and r/m fields in the second byte of the instruction. As an example, we consider a simple move instruction and the various ways in which memory can be addressed.

The instruction we consider is MOVE immediate operand to memory. As we see above, MOVE M, DATA is represented by:

C6 XX DATA

or

C7 XX DATA DATA

where XX is the hexadecimal representation of the second byte of the instruction, and the field labelled "DATA" represents one byte of arbitrary data. We now consider the various possibilities.

If mod = 00, and r/m = 110, then the EA is given by disp-high:disp-low. The second byte of the instruction is:

mod 000 r/m

or

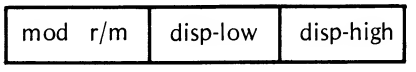
00 000 110

or

06

in hexadecimal representation.

When a displacement is used in the effective address, the displacement is given in the third and fourth bytes of the instruction, that is,



The use of the notation x:y, where x and y are fields, must be explained. The x field denotes the 16-bit segment address value, and the y field denotes the 16-bit offset address value. The effective address (EA) is then calculated by displacing the segment address by four low-order 0 bits and adding the offset address to the displaced value.

An illustrative example would be helpful at this point. Consider the instruction:

C6 06 00 02 77

From the above analysis, we know that C6 represents a MOVE M, DATA instruction. The second byte 06 indicates that mod = 00 and r/m = 110, that is, the EA is given by disp-high:disp-low.

The third and fourth bytes of the instruction therefore contain the displacement, the third byte being the low-order value and the fourth byte being the high-order

TABLE 4.2. Intel 8086 Instruction Set Summary[a]

DATA TRANSFER

MOV = Move:	76543210	76543210	76543210	76543210
Register/memory to/from register	100010dw	mod reg r/m		
Immediate to register/memory	1100011w	mod000 r/m	data	data if w=1
Immediate to register	1011w reg	data	data if w=1	
Memory to accumulator	1010000w	addr-low	addr-high	
Accumulator to memory	1010001w	addr-low	addr-high	
Register/memory to segment register	10001110	mod0reg r/m		
Segment register to register/memory	10001100	mod0reg r/m		

PUSH = Push:		
Register/memory	11111111	mod110 r/m
Register	01010 reg	
Segment register	000reg110	

POP = Pop:		
Register/memory	10001111	mod000 r/m
Register	01011 reg	
Segment register	000reg111	

XCHG = Exchange:		
Register/memory with register	1000011w	mod reg r/m
Register with accumulator	10010 reg	

IN/INW = Input to AL/AX from:		
Fixed port	1110010w	port
Variable port	1110110w	

OUT/OUTW = Output from AL/AX to:		
Fixed port	1110011w	port
Variable port	1110111w	
XLAT=Translate byte to AL	11010111	
LEA=Load EA to register	10001101	mod reg r/m
LDS=Load pointer to DS	11000101	mod reg r/m
LES=Load pointer to ES	11000100	mod reg r/m
LAHF=Load AH with flags	10011111	
SAHF=Store AH into flags	10011110	
PUSHF=Push flags	10011100	
POPF=Pop flags	10011101	

ARITHMETIC

ADD = Add:				
Reg./memory with register to either	000000dw	mod reg r/m		
Immediate to register/memory	100000sw	mod000 r/m	data	data if s:w=01
Immediate to accumulator	0000010w	data	data if w=1	

ADC = Add with carry:				
Reg./memory with register to either	000100dw	mod reg r/m		
Immediate to register/memory	100000sw	mod010 r/m	data	data if s:w=01
Immediate to accumulator	0001010w	data	data if w=1	

INC = Increment:		
Register/memory	1111111w	mod000 r/m
Register	01000 reg	
AAA=ASCII adjust for add	00110111	
DAA=Decimal adjust for add	00100111	

SUB = Subtract:				
Reg./memory and register to either	001010dw	mod reg r/m		
Immediate from register/memory	100000sw	mod101 r/m	data	data if s:w=01
Immediate from accumulator	0010110w	data	data if w=1	

SBB = Subtract with borrow				
Reg./memory and register to either	000110dw	mod reg r/m		
Immediate from register/memory	100000sw	mod011 r/m	data	data if s:w=01
Immediate from accumulator	0001110w	data	data if w=1	

DEC = Decrement:	76543210	76543210	76543210	76513210
Register/memory	1111111w	mod001 r/m		
Register	01001 reg			
NEG=Change sign	1111011w	mod011 r/m		

CMP = Compare:				
Register/memory and register	001110dw	mod reg r/m		
Immediate with register/memory	100000sw	mod111 r/m	data	data if s:w=01
Immediate with accumulator	0011110w	data	data if w=1	
AAS=ASCII adjust for subtract	00111111			
DAS=Decimal adjust for subtract	00101111			
MUL=Multiply (unsigned)	1111011w	mod100 r/m		
IMUL=Integer multiply (signed)	1111011w	mod101 r/m		
AAM=ASCII adjust for multiply	11010100	00001010		
DIV=Divide (unsigned)	1111011w	mod110 r/m		
IDIV=Integer divide (signed)	1111011w	mod111 r/m		
AAD=ASCII adjust for divide	11010101	00001010		
CBW=Convert byte to word	10011000			
CWD=Convert word to double word	10011001			

LOGIC

NOT=Invert	1111011w	mod010 r/m		
SHL/SAL=Shift logical/arithmetic left	110100vw	mod100 r/m		
SHR=Shift logical right	110100vw	mod101 r/m		
SAR=Shift arithmetic right	110100vw	mod111 r/m		
ROL=Rotate left	110100vw	mod000 r/m		
ROR=Rotate right	110100vw	mod001 r/m		
RCL=Rotate through carry flag left	110100vw	mod010 r/m		
RCR=Rotate through carry right	110100vw	mod011 r/m		

AND = And:				
Reg./memory and register to either	001000dw	mod reg r/m		
Immediate to register/memory	1000000w	mod100 r/m	data	data if w=1
Immediate to accumulator	0010010w	data	data if w=1	

TEST = And function to flags, no result:				
Register/memory and register	1000010w	mod reg r/m		
Immediate data and register/memory	1111011w	mod000 r/m	data	data if w=1
Immediate data and accumulator	1010100w	data	data if w=1	

OR = Or:				
Reg./memory and register to either	000010dw	mod reg r/m		
Immediate to register/memory	1000000w	mod001 r/m	data	data if w=1
Immediate to accumulator	0000110w	data	data if w=1	

XOR = Exclusive or:				
Reg./memory and register to either	001100dw	mod reg r/m		
Immediate to register/memory	1000000w	mod110 r/m	data	data if w=1
Immediate to accumulator	0011010w	data	data if w=1	

STRING MANIPULATION

REP=Repeat	1111001z
MOVB/MOVW=Move byte/word	1010010w
CMPB/CMPW=Compare byte/word	1010011w
SCAB/SCAW=Scan byte/word	1010111w
LODB/LODW=Load byte/wd to AL/AX	1010110w
STOB/STOW=Stor byte/wd frm AL/A	1010101w

Mnemonics ©Intel, 1978

[a] *Reprinted by permission, copyright 1978, Intel Corporation.*

TABLE 4.2. (*Continued*)

CONTROL TRANSFER

CALL = Call:	76543210	76543210	76543210
Direct within segment	11101000	disp-low	disp-high
Indirect within segment	11111111	mod 010 r/m	
Direct intersegment	10011010	offset-low	offset-high
		seg-low	seg-high
Indirect intersegment	11111111	mod 011 r/m	

JMP = Unconditional Jump:			
Direct within segment	11101001	disp-low	disp-high
Direct within segment-short	11101011	disp	
Indirect within segment	11111111	mod 100 r/m	
Direct intersegment	11101010	offset-low	offset-high
		seg-low	seg-high
Indirect intersegment	11111111	mod 101 r/m	

RET = Return from CALL:			
Within segment	11000011		
Within seg. adding immed to SP	11000010	data-low	data-high
Intersegment	11001011		
Intersegment, adding immediate to SP	11001010	data-low	data-high

JE/JZ=Jump on equal/zero	01110100	disp
JL/JNGE=Jump on less/not greater or equal	01111100	disp
JLE/JNG=Jump on less or equal/not greater	01111110	disp
JB/JNAE=Jump on below/not above or equal	01110010	disp
JBE/JNA=Jump on below or equal/not above	01110110	disp
JP/JPE=Jump on parity/parity even	01111010	disp
JO=Jump on overflow	01110000	disp
JS=Jump on sign	01111000	disp
JNE/JNZ=Jump on not equal/not zero	01110101	disp
JNL/JGE=Jump on not less/greater or equal	01111101	disp
JNLE/JG=Jump on not less or equal/greater	01111111	disp

	76543210	76543210
JNB/JAE=Jump on not below/above or equal	01110011	disp
JNBE/JA=Jump on not below or equal/above	01110111	disp
JNP/JPO=Jump on not par/par odd	01111011	disp
JNO=Jump on not overflow	01110001	disp
JNS=Jump on not sign	01111001	disp
LOOP=Loop CX times	11100010	disp
LOOPZ/LOOPE=Loop while zero/equal	11100001	disp
LOOPNZ/LOOPNE=Loop while not zero/equal	11100000	disp
JCXZ=Jump on CX zero	11100011	disp

INT = Interrupt		
Type specified	11001101	type
Type 3	11001100	
INTO=Interrupt on overflow	11001110	
IRET=Interrupt return	11001111	

PROCESSOR CONTROL

CLC=Clear carry	11111000	
CMC=Complement carry	11110101	
STC=Set carry	11111001	
CLD=Clear direction	11111100	
STD=Set direction	11111101	
CLI=Clear interrupt	11111010	
STI=Set interrupt	11111011	
HLT=Halt	11110100	
WAIT=Wait	10011011	
ESC=Escape (to external device)	11011 x	mod x r/m
LOCK=Bus lock prefix	11110000	

Footnotes:

AL = 8-bit accumulator
AX = 16-bit accumulator
CX = Count register
DS = Data segment
ES = Extra segment
Above/below refers to unsigned value.
Greater = more positive;
Less = less positive (more negative) signed values
if d = 1 then "to"; if d = 0 then "from"
if w = 1 then word instruction; if w = 0 then byte instruction

if mod = 11 then r/m is treated as a REG field
if mod = 00 then DISP = 0*, disp-low and disp-high are absent
if mod = 01 then DISP = disp-low sign-extended to 16-bits, disp-high is absent
if mod = 10 then DISP = disp-high: disp-low
if r/m = 000 then EA = (BX) + (SI) + DISP
if r/m = 001 then EA = (BX) + (DI) + DISP
if r/m = 010 then EA = (BP) + (SI) + DISP
if r/m = 011 then EA = (BP) + (DI) + DISP
if r/m = 100 then EA = (SI) + DISP
if r/m = 101 then EA = (DI) + DISP
if r/m = 110 then EA = (BP) + DISP*
if r/m = 111 then EA = (BX) + DISP
DISP follows 2nd byte of instruction (before data if required)

*except if mod = 00 and r/m = 110 then EA = disp-high: disp-low.

if s:w = 01 then 16 bits of immediate data form the operand.
if s:w = 11 then an immediate data byte is sign extended to form the 16-bit operand.
if v = 0 then "count" = 1; if v = 1 then "count" in (CL)
x = don't care
z is used for string primitives for comparison with ZF FLAG.

SEGMENT OVERRIDE PREFIX

001 reg 110

REG is assigned according to the following table:

16-Bit (w = 1)	8-Bit (w = 0)	Segment
000 AX	000 AL	00 ES
001 CX	001 CL	01 CS
010 DX	010 DL	10 SS
011 BX	011 BL	11 DS
100 SP	100 AH	
101 BP	101 CH	
110 SI	110 DH	
111 DI	111 BH	

Instructions which reference the flag register file as a 16-bit object use the symbol FLAGS to represent the file:

FLAGS = X:X:X:X:(OF):(DF):(IF):(TF):(SF):(ZF):X:(AF):X:(PF):X:(CF)

Mnemonics © Intel, 1978

value. The low-order value is therefore 00, and the high-order value is 02.

For mod = 00 and r/m = 110, the EA does *not* simply adjoin the low- and high-order values to produce a single displacement (DISP) value. Rather, the effective address is given by disp-high:disp-low, in our case, 02:00.

Let us calculate 02:00. First, we express these hexadecimal values in binary representation so that the calculation is fully understood. The first field, 02, is the 16-bit segment address value:

0000 0000 0000 0010

and the second byte is the 16-bit offset value:

0000 0000 0000 0000

It should be noted that in our displacement representation, the high-order zero fields are suppressed for simplicity. The correct representation should actually be

0002:0000

A more complex displacement is considered in a later example.

Continuing our calculation of the EA, the segment address is first displaced by four low-order 0 bits:

0000 0000 0000 0010 0000

and adding the offset address

0000 0000 0000 0000

thereto, we arrive at the 20-bit effective address:

0000 0000 0000 0010 0000

In hexadecimal representation:

00020

The instruction is not yet finished. The DATA field is the last field of the instruction, which is 77. Thus the value 77 is placed in the memory byte represented by address 00020.

Let us now turn to other addressing modes.

mod = 00

The exceptional case of mod = 00 and r/m = 110 is considered above. When r/m is not equal to 110, then disp-high and disp-low are absent from the instruction, and DISP = 00. The EA address is calculated solely from the values contained in the registers in the 8086. Which registers are used is determined from the r/m field:

If r/m = 000, EA = (BX) + (SI)

where (BX) represents the contents of the 16-bit base register BX, and (SI) represents the contents of the 16-bit index register SI.

If r/m = 001, EA = (BX) + (DI)

If r/m = 010, EA = (BP) + (SI)

If r/m = 011, EA = (BP) + (DI)

If r/m = 100, EA = (SI)

If r/m = 101, EA = (DI)

If r/m = 111, EA = (BX)

We now write a simple program using an alternative technique of calculating the EA. Let us use, for example, the BX register. From the data above, r/m should be set equal to 111. Thus the second byte of the instruction should be

00 000 111

or

07

in hexadecimal representation.

Consider again the instruction MOV M, DATA, where M is now determined by the BX register. The instruction in hexadecimal notation is

C6 07 DATA

where DATA is the 1-byte data field. Let us use DATA = 77 as the example. Then the complete instruction is:

C6 07 77

Note that unlike the previous example, the displacement field is absent from the instruction. There is no displacement in the instruction in this example.

Let us consider this instruction MOV M, DATA somewhat further and specify the value of the BX register so that the DATA is actually moved to a desired location.

To set the BX register, we use the MOV BX, DATA, DATA instruction. The location we choose as 0200 in hexadecimal, that is, the same as the location 20:0 using the displacement notation.

The instruction MOV BX data-low, data-high is:

BB 00 02

where data-low = 00 and data-high = 02.

Let us now consider the complete program to first set the BX register, and then move the data field 77 to the location specified by the BX register.

```
MOV  BX  00  02
MOV  M(BX)  77
```

or in hexadecimal notation:

```
BB  00  02
C6  07  77
```

When these two instructions are executed, the value 77 is placed in the memory location 0200.

We now turn to other addressing modes.

mod = 01

When mod = 01, a 1-byte displacement is used with the instruction:

op code	mod r/m	disp-low

The displacement is equal to the disp-low, extended to 16 bits. The effective address (EA) is calculated:

If r/m = 000, EA = (BX) + (SI) + DISP
If r/m = 001, EA = (BX) + (DI) + DISP
If r/m = 010, EA = (BP) + (SI) + DISP
If r/m = 011, EA = (BP) + (DI) + DISP
If r/m = 100, EA = (SI) + DISP
If r/m = 101, EA = (DI) + DISP
If r/m = 110, EA = (BP) + DISP
If r/m = 111, EA = (BX) + DISP

We again consider the example of moving data to a specified memory location. Let us now pick the location 00FF. We note that the high-order byte is 00, and thus we can use the mod = 01 addressing technique.

The second byte of the instruction is:

0 1 0 0 0 1 1 1 or 47 in hexadecimal notation

We note that mod = 01, and r/m = 1 1 1, so that the effective address is (BX) + DISP.

There are a number of alternative ways to move the data to memory location 00FF now that we have *both* the BX register and the DISP field to use. First we use the BX register alone, that is set DISP = 0000. The BX register is then set to the value 00FF by the instruction

MOV BX FF 00

or

BB FF 00

Next, we move the data field 77 with the instruction

MOV M(BX), 77

or

C6 47 00 77

Thus the value 77 is moved to the location specified by (BX) + 0000. Since BX = 00FF, the location of the effective address is 00FF.

Another method is to use the DISP field alone. First we set the BX register to 0000,

MOV BX 00 00

or

BB 00 00

Next we use the MOV M(BX) instruction, with DISP = 00FF,

MOV M(BX) FF 77

or

C6 47 FF 77

Here the disp-low field is FF, the disp-high field is absent, and the DATA field is 77.

Finally, we use the BX register together with the DISP field. Let us choose the memory location destination as 0200 hexadecimal. We set BX = 0101 and DISP = 00FF. It is evident that (BX) + DISP = 0101 + 00FF = 0200.

The sequence of instructions would be:

MOV BX, 01 01 or BB 01 01
MOV M(BX) DISP 77 C6 47 FF 77

The result is that the data field 77 is placed in memory location 0200.

mod = 10

When mod = 10, the 2-byte displacement field is used with the instruction:

op code	mod r/m	disp-low	disp-high

and the displacement is given by

DISP = disp-high:disp-low

We again consider the example of MOV M(BX) DATA. The second byte of the instruction is

1 0 0 0 0 1 1 1 or 87 in hexadecimal notation

We note that mod = 10, and r/m = 111, so the EA is given by

EA = (BX) + DISP
= (BX) + disp-high:disp-low

Let us again take the problem of moving the data field 77 to memory location 0200, or 20:0 in the segment: offset notation.

We first set the BX register to 0000 by the instruction

MOV BX 00 00 or BB 00 00

Then we move the data field 77 to the desired memory location by

MOV M(BX) DISP 77 or C6 87 00 02 77

We note that disp-low = 00, and disp-high = 02. Thus

disp-high:disp-low is 02:00. The effective address (EA) is therefore

$$\begin{aligned}(BX) + \text{disp-high:disp-low} &= 0000 + 02{:}00\\ &= 0000\ \ 0000\\ &+ 0010\ \ 0000\\ &+ 0000\ \ 0000\\ &\overline{0010\ \ 0000}\end{aligned}$$

Another method is to use the BX register alone and let DISP = 00 00. The sequence of instructions is:

```
MOV  BX  00  02
MOV  M(BX)  DISP  77
```

or

```
BB  00  02
C6  87  00  00  77
```

The EA is computed using BX, and the data field 77 is moved to memory location 0200, or 20:0 in segment: offset notation.

One can also use the BX register together with the displacement. The following program performs the same data movement as that above:

```
BB  FF  01
C6  87  01  00  77
```

It is noted that 01FF + 0001 = 0200.

The general form of the second byte of the instruction

```
mod  reg  r/m
```

may be computed. Consider mod = 00, and r/m = 111, so EA = (BX). The possibilities for the different registers are:

AX	07
CX	0F
DX	17
BX	1F
SP	27
BP	2F
SI	37
DI	3F

assuming that w = 1. If w = 0 the values above are the same for the second byte of the instruction, but the registers referred to are

AL
CL
DL
BL
AH
CH
DH
BH

The second byte may also be used with register operands. In this case both reg and r/m designate registers. The reg field denotes the source, and the r/m field denotes the destination, when d = 0 in the first byte of the instruction. The coding is as follows:

Destination	*Source*	*Code*
AX	AX	C0
CX	AX	C8
DX	AX	D0
BX	AX	D8
SP	AX	E0
BP	AX	E8
SI	AX	F0
DI	AX	F8
AX	CX	C1
CX	CX	C9
DX	CX	D1
BX	CX	D9
SP	CX	E1
BP	CX	E9
SI	CX	F1
DI	CX	F9
AX	DX	C2
CX	DX	CA
DX	DX	D2
BX	DX	DA
SP	DX	E2
BP	DX	EA
SI	DX	F2
DI	DX	FA
AX	BX	C3
CX	BX	CB
DX	BX	D3
BX	BX	DB
SP	BX	E3
BP	BX	EB
SI	BX	F3
DI	BX	FB
AX	SP	C4
CX	SP	CC
DX	SP	D4
BX	SP	DC
SP	SP	E4
BP	SP	EC
SI	SP	F4
DI	SP	FC
AX	BP	C5
CX	BP	CD
DX	BP	D5

Destination	*Source*	*Code*
BX	BP	DD
SP	BP	E5
BP	BP	ED
SI	BP	F5
DI	BP	FD
AX	SI	C6
CX	SI	CE
DX	SI	D6
BX	SI	DE
SP	SI	E6
BP	SI	EE
SI	SI	F6
DI	SI	FE
AX	DI	C7
CX	DI	CF
DX	DI	D7
BX	DI	DF
SP	DI	E7
BP	DI	EF
SI	DI	F7
DI	DI	FF

Use of Segment Override Prefix

Consider the instruction

26 8A 08

The instruction 8A is recognized as a MOVE instruction, and the 26 is a segment override prefix. The general form of the segment override prefix is:

0 0 0 reg 1 1 0

The segment overide prefixes are therefore:

Segment	*Prefix*
ES	26
CS	2E
SS	36
DS	3E

The effect of the prefix is that the segment register selected by the prefix is used to compute the physical address of the instruction that follows the prefix.

In the example given above, the 08 byte would specify that the effective address is normally (BX) + (SI), and that the data are moved to the CL register. Now, however, the ES address is used.

The Detailed 8086 Instruction Set

To facilitate the understanding and use of the 8086 instruction set in hexadecimal notation, we express the op codes of a sample of the instructions in hexadecimal form.

The convention of the mnemonics used in the assembly language should carefully be noted. The instruction MOV AX ES, means that the content of the ES register be moved *to* the AX register. Similarly, MOV AL, DATA means that the DATA byte is to be moved to the AL register.

A portion of the 8086 instruction set in hexadecimal notation follows.*

MOVE

register operand to memory

MOV AL	88 00	(assuming mod = 00, r/m = 000)
MOV CL	88 08	
MOV DL	88 10	
MOV BL	88 18	
MOV AH	88 20	
MOV CH	88 28	
MOV DH	88 30	
MOV BH	88 38	
MOV AX	89 00	
MOV CX	89 08	
MOV DX	89 10	
MOV BX	89 18	
MOV SP	89 20	
MOV BP	89 28	
MOV SI	89 30	
MOV DI	89 38	

immediate operand to memory

MOV M, DATA	C6 00 DATA
MOV M, DATA, DATA	C7 00 DATA DATA

immediate operand to register

MOV AL, DATA	B0 DATA
MOV CL, DATA	B1 DATA
MOV DL, DATA	B2 DATA
MOV BL, DATA	B3 DATA
MOV AH, DATA	B4 DATA
MOV CH, DATA	B5 DATA
MOV DH, DATA	B6 DATA
MOV BH, DATA	B7 DATA
MOV AX, DATA, DATA	B8 DATA DATA
MOV CX, DATA, DATA	B9 DATA DATA
MOV DX, DATA, DATA	BA DATA DATA

*Mnemonics © 1978, Intel Corp.

MOV BX, DATA, DATA BB DATA DATA
MOV SP, DATA, DATA BC DATA DATA
MOV BP, DATA, DATA BD DATA DATA
MOV SI, DATA, DATA BE DATA DATA
MOV DI, DATA, DATA BF DATA DATA

memory operand to accumulator
MOV, ADDR-LOW, ADDR-HIGH A0 (to AL)
MOV, ADDR-LOW, ADDR-HIGH A1 (to AX)

accumulator to memory operand
MOV, ADDR-LOW, ADDR-HIGH A2 (from AL)
MOV, ADDR-LOW, ADDR-HIGH A3 (from AX)

memory operand to segment register
MOV ES, M 8E 00 (assuming mod = 00, r/m = 000)
MOV CS, M 8E 08
MOV SS, M 8E 10
MOV DS, M 8E 18

register operand to segment register
MOV ES, AX 8E C0

segment register to memory
MOV M, ES 8C 00
MOV M, CS 8C 08
MOV M, SS 8C 10
MOV M, DS 8C 18

segment register to register
MOV AX, ES 8C C0
MOV CX, ES 8C C1
MOV DX, ES 8C C2
MOV BX, ES 8C C3
MOV SP, ES 8C C4
MOV BP, ES 8C C5
MOV SI, ES 8C C6
MOV DI, ES 8C C7
MOV AX, CS 8C C8
MOV CX, CS 8C C9
MOV DX, CS 8C CA
MOV BX, CS 8C CB
MOV SP, CS 8C CC
MOV BP, CS 8C CD
MOV SI, CS 8C CE
MOV DI, CS 8C CD

PUSH

memory operand
PUSH M FF 30

register operand
PUSH AX 50
PUSH CX 51
PUSH DX 52
PUSH BX 53
PUSH SP 54
PUSH BP 55
PUSH SI 56
PUSH DI 57

segment register
PUSH ES 06
PUSH CS 0E
PUSH SS 16
PUSH DS 1E

POP

memory operand
POP M 8F 30

register operand
POP AX 58
POP CX 59
POP DX 5A
POP BX 5B
POP SP 5C
POP BP 5D
POP SI 5E
POP DI 5F

segment register
POP ES 07
POP CS 0F
POP SS 17
POP DS 1F

XCHG

memory operand with register operand
XCHG AL 86 00 (assuming mod = 00, r/m = 000)
XCHG CL 86 08
XCHG DL 86 10
XCHG BL 86 18
XCHG AH 86 20
XCHG CH 86 28
XCHG DH 86 30

XCHG BH 86 38
XCHG AX 87 00
XCHG CX 87 08
XCHG DX 87 10
XCHG BX 87 18
XCHG SP 87 20
XCHG BP 87 28
XCHG SI 87 30
XCHG DI 87 38

register operand with register operand
XCHG AL, CL 86 C1

IN Byte or INW Word

fixed port
IN (port) E8 (port)
INW (port) E9 (port)

variable port
IN EC
INW ED

The source is defined by (DX).

OUT Byte and OUTW Word

fixed port
OUT (port) E6 (port)
OUTW (port) E7 (port)

variable port
OUT EE
OUTW EF

XLAT

table lookup byte translation
XLAT D7

LEA

load effective address to defined register
LEA 8D XX

LDS

load pointer into DS; that is, a segment: offset pair is transferred to the DS register and a specified register
LDS C5 XX

LES

load pointer into ES; that is, a segment: offset pair is transferred into the ES register and a specified register
LES C4 XX

LAHF

load AH with flags
LAHF 9F

SAHF

transfer contents of the AH register to the flag register
SAHF 9E

PUSHF

decrements SP by 2 and transfer flag register to stack element addressed by SP
PUSHF 9C

POPF

transfers stack element addressed by SP to flag register and increments SP by 2
POPF 9D

ADD

memory with register operand, or register operand with register operand

1. destination: register
ADD 02 XX (byte)
ADD 03 XX (word)

2. destination: effective address
ADD 00 XX (byte)
ADD 01 XX (word)

immediate operand to memory or register operand
ADD XX DATA 80 XX DATA
ADD XX DATA DATA 81 XX DATA DATA
ADD XX DATA 82 XX DATA
83 XX DATA (where DATA is sign extended to form 16-bit operand)

immediate operand to accumulator
ADD AL, DATA 04 DATA
ADD AX, DATA 05 DATA DATA

ADC

add with carry

memory with register operand, or register operand with register operand

1. destination: register
ADC (reg) 12 XX (byte)
13 XX (word)

2. destination: effective address
ADC (addr) 10 XX (byte)
11 XX (word)

immediate operand to memory or register operand

ADC XX DATA	80 XX DATA
ADC XX DATA DATA	81 XX DATA DATA
ADC XX DATA	82 XX DATA
ADC XX DATA	83 XX DATA (where DATA is sign extended to form a 16-bit operand)

immediate operand to accumulator

ADC AL, DATA	14 DATA
ADC AX, DATA	15 DATA

INC

adds 1 to the source operand

memory or register operand

INC (addr)	FE XX (byte)
INC (addr)	FF XX (word)
INC AL	FE C0
INC BL	FE C1
INC DL	FE C2
INC BL	FE C3
INC AH	FE C4
INC CH	FE C5
INC DH	FE C6
INC BH	FE C7

register operand

INC AX	40
INC CX	41
INC DX	42
INC BX	43
INC SP	44
INC BP	45
INC SI	46
INC DI	47

AAA

unpacked BCD (ASCII) adjust addition

AAA	37

DAA

decimal adjust for addition

DAA	27

SUB

subtract

memory with register operand, or register operand with register operand

1. destination: register

SUB (reg), (addr)	2A XX	(byte)
	2B XX	(word)

2. destination: effective address

SUB (addr), (reg)	28 XX	(byte)
	29 XX	(word)

immediate operand from memory

SUB (addr), DATA	80 XX DATA
SUB (addr), DATA DATA	81 XX DATA DATA
SUB (addr), DATA	82 XX DATA
SUB (addr), data	83 XX DATA (DATA is sign extended)

immediate operand from accumulator

SUB AL, DATA	2C DATA
SUB AX, DATA DATA	2D DATA DATA

SBB

subtract with borrow

memory from register operand, or operand from register operand

1. destination: register

SBB (reg), (addr)	1A XX (byte)
	1B XX (word)

2. destination: effective address

SBB (addr), (reg)	18 XX (byte)
	19 XX (word)

immediate operand from memory or register operand

SBB (addr or reg), DATA	80 XX DATA
	81 XX DATA DATA
	82 XX DATA
	83 XX DATA

We note that the first byte of the instruction is the same as for the ADD, ADC, and SUB instructions. Thus we can be more specific in our listing for this instruction:

SBB (BX), DATA	80 1F DATA
SBB AL, DATA	80 D8 DATA
SBB CL, DATA	80 D9 DATA

immediate operand from accumulator

SBB AL, DATA	1C DATA
SBB AX, DATA	1D DATA DATA

DEC

decrement 1 from source operand

memory or register operand

DEC (addr)	FE XX (byte)

DEC (addr)	FF XX (word)
DEC AL	FE C8
DEC CL	FE C9
DEC DL	FE CA
DEC BL	FE CB
DEC AH	FE CC
DEC CH	FE CD
DEC DH	FE CE
DEC BH	FE CF
DEC AX	FFC8
DEC CX	FF C9
DEC DX	FF CA
DEC BX	FF CB
DEC SP	FF CC
DEC BP	FF CD
DEC SI	FF CE
DEC DI	FF CF

NEG

subtracts source operand from 0

NEG	F6 XX (byte)
	F7 XX (word)

CMP

compare, performs subtraction to affect flags

memory operand or register operand with register operand

CMP	38 XX (byte)
	39 XX (word)
	3A XX (byte)
	3B XX (word)

immediate operand with memory or data operand

CMP	80 XX DATA
	81 XX DATA DATA
	82 XX DATA
	83 XX DATA

immediate operand with accumulator

CMP	3C DATA
	3D DATA DATA

AAS

ASCII adjust for subtraction

AAS	3F

5

Zilog Z8000 system design

The Zilog Z8000 is another very important 16-bit microprocessor introduced slightly later than the Intel 8086, and it would also be worthwhile examining the operation and system design using the Z8000 in greater detail. In the present chapter we consider the following features:

- Internal architecture.
- System configuration.
- System timing.
- Instruction set.

Internal Architecture

As we point out in Chapter 2, the Zilog Z8000 is a 16-bit *n*-channel microprocessor available in either 48-pin segmented CPU package, or a 40-pin nonsegmented CPU package. The 48 pin version is called the Z8001, while the 40 pin version is called the Z8002. A pin diagram of both versions is shown in Figure 5.1. The definition of the pins is given in Table 5.1.

The basic difference between the Z8001 and Z8002 is the addressing capability of the device. The Z8002 directly addresses 64 kilobytes, while the Z8001 addresses 8 megabytes of memory. Memory extension is possible through the use of internal modes so that up to 48 megabytes may be addressed by the Z8001 and up to 384 kilobytes by the Z8002.

The general-purpose registers of the Z8000 are shown in Figures 5.2 and 5.3, where Figure 5.2 refers to the Z8001 and Figure 5.3 to the Z8002. A brief explanation of the registers, with reference to Figures 5.2 and 5.3, is necessary at this point.

There are basically sixteen 16-bit general-purpose registers in the Z8000 labeled RO through R15. For byte operations, the first eight registers R0 through R7 are treated as 8-bit registers and are labeled with high (H) or low (L) designations, for example, the 16-bit register R0 is formed by two 8-bit registers RH0 and RL0.

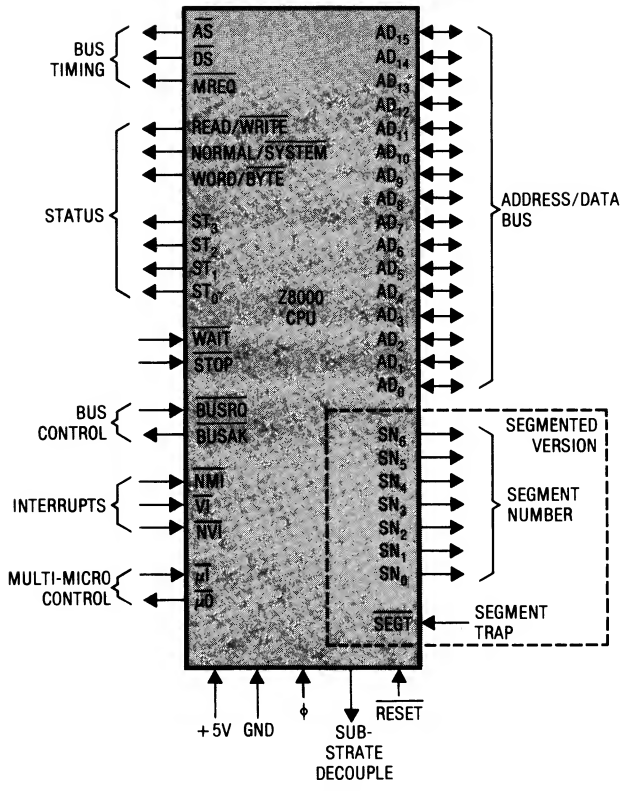


FIGURE 5.1 Zilog Z8000 pin diagram. Reproduced by permission. © 1979 Zilog, Inc. This material shall not be reproduced without the written consent of Zilog, Inc.

There are also two other methods of grouping the registers: in pairs and in quadruples. In pairs the registers form 32-bit words and are labeled RR0 through RR14. In quadruples the registers form 64-bit words and are labeled RQ0, RQ4, RQ8, and RQ12.

As far as the register organization we have described so far is concerned the Z8001 and Z8002 are identical. The difference arises in the implied definition of the R14 and R15 registers.

In the Z8001 the register pair RR14 forms the implied stack pointer. The register R14 contains a 7-bit segment number, and R15 contains the 16-bit offset. Further details concerning the segment:offset technique of addressing memory are presented later. In the Z8002, the register R14 is a general-purpose register, while R15 is the implied 16-bit stack pointer.

Common to both the Z8001 and the Z8002 is a refresh counter shown in Figure 5.4. The refresh counter consists of a single enable bit (RE), a 6-bit rate counter, and a 9-bit row counter. The purpose of the counter is to automatically refresh dynamic memory at a rate that is programmable between 1 and 64 μsec. The 9-bit row counter functions to address up to 256 rows of memory. The rate counter determines the rate of refresh, that is, the time between successive refreshes, and is driven at one-fourth the CPU clock rate. The refresh register may be set by a Load Control Register Refresh instruction to define appropriate rate and counter field contents, which defines how the memory is refreshed.

The final important group of registers the program status registers shown in Figure 5.5 for the Z8001 and in Figure 5.6 for the Z8002.

In the Z8002 the program status group consists of two words, the program counter (PC) and the flag and control word (FCW). In the Z8001 the program status group consists additionally of a reserved word for future use and another word in the program counter and status area pointer associated with the segment:offset nature of the addressing in the Z8001.

The fields in the FCW shown in Figure 5.7, are defined as follows:

SEG	Segment
S/N	System or not-normal mode
VIE	Vector interrupt enable
NVIE	Nonvectored interrupt enable
C	Carry
Z	Zero
S	Sign
P/V	Parity/overflow
DA	Decimal adjust
H	Half-carry flag

One of the important distinctions between the Intel 8086 and the Zilog Z8000 is the interrupt and trap structure of the latter. The distinction between an interrupt and a trap can be made here. An interrupt is an external asynchronous event. A trap is a synchronous event that is internal to the system. Both interrupts and traps are processed in the same manner by the CPU.

The Z8000 is able to handle three types of interrupts:

- Nonmaskable.
- Vectored.
- Nonvectored.

Both vectored and the nonvectored interrupts are maskable. There is a control instruction called Disable Interrupt (DI) that disables either the vectored or the nonvectored interrupts. The masking procedure may be reversed by the Enable Interrupt (EI) instruction. The nonmaskable interrupt is highest priority and is not affected by these instructions.

The Z8000 also supports four traps:

- System call.
- Unimplemented instruction.

TABLE 5.1. Zilog Z8000 Pin Definitions[a]

Pin Description

AD_0-AD_{15}. *Address/Data* (inputs/outputs, active High, 3-state). These multiplexed address and data lines are used both for I/O and to address memory.

$\overline{AS}$. *Address Strobe* (output, active Low, 3-state). The rising edge of $\overline{AS}$ indicates addresses are valid.

$\overline{BUSAK}$. *Bus Acknowledge* (output, active Low). A Low on this line indicates the CPU has reliquished control of the bus.

$\overline{BUSRQ}$. *Bus Request* (input, active Low). This line must be driven Low to request the bus from the CPU.

$\overline{DS}$. *Data Strobe* (output, active Low, 3-state). This line times the data in and out of the CPU.

$\overline{MREQ}$. *Memory Request* (output, active Low, 3-state). A Low on this line indicates that the address/data bus hold a memory address.

M_I, M_O. *Multi-Micro In, Multi-Micro Out* (input and output, active Low). These two lines form a resource-request daisy chain that allows one CPU in a multi-microprocessor system to access a shared resource.

$\overline{NMI}$. *Non-Maskable Interrupt* (edge triggered, input, active Low). A high-to-low transition on $\overline{NMI}$ requests a non-maskable interrupt. The NMI interrupt has the highest priority of the three types of interrupts.

$\overline{NVI}$. *Non-Vectored Interrupt* (input, active Low). A Low on this line requests a non-vectored interrupt.

CLK. *System Clock* (input). CLK is a 5V single-phase time-base input.

$\overline{RESET}$. *Reset* (input, active Low). A Low on this line resets the CPU.

R/$\overline{W}$. *Read/Write* (output, Low = Write, 3-state). R/$\overline{W}$ indicates that the CPU is reading from or writing to memory or I/O.

SN_0-SN_6. *Segment Number* (outputs, active High, 3-state). These lines provide the 7-bit segment number used to address one of 128 segments by the Z8010 Memory Management Unit. Output by the Z8001 only.

$\overline{SEGT}$. *Segment Trap* (input, active Low). The Memory Management Unit interrupts the CPU with a Low on this line when the MMU detects a segmentation trap.

ST_0-ST_3. *Status* (outputs, active High, 3-state). These lines specify the CPU status (see table).

$\overline{STOP}$. *Stop* (input, active Low). This input can be used to single-step instruction execution.

$\overline{VI}$. *Vectored Interrupt* (input, active Low). A Low on this line requests a vectored interrupt.

$\overline{WAIT}$. *Wait* (input, active Low). This line indicates to the CPU that the memory or I/O device is not ready for data transfer.

B/$\overline{W}$. *Byte/Word* (output, Low = Word, 3-state). This signal defines the type of memory reference on the 16-bit address/data bus.

N/$\overline{S}$. *Normal/System Mode* (output, Low = System Mode, 3-state). N/$\overline{S}$ indicates the CPU is in the normal or system mode.

Decouple. Output from on-chip negative substrate-bias generator. Presently not connected.

[a] *Reproduced by permission. © 1979 Zilog, Inc. This material shall not be reproduced without the written consent of Zilog, Inc.*

- Privileged instruction.
- Segmentation trap.

The first three traps are known as internal traps and are of highest priority, even above that of the nonmaskable interrupt. The last trap, the segmentation trap, is an external trap generated by a support chip, the Z8010.

The traps are associated with the two types of operational modes of the Z8000: the system mode and the normal mode. These modes are distinguished by the existence of privileged instructions that operate in one mode and not the other. The use of such modes is important in the implementation of operating systems since it simplifies the programming task considerably in instances where one wishes to define "system" areas in memory that cannot be changed by the user.

A trap occurs when a "privileged" instruction (as is defined and listed in a subsequent section of this chapter) that is limited to the system mode is used in the normal mode. A trap may also occur by a system call, or by an unimplemented instruction.

The effect of a trap on the system is the same as that of an interrupt. The current program status is automatically pushed on the system stack, together with a 16-bit identifier indicating the reason or source of the trap or interrupt. For example, for an internal trap, the

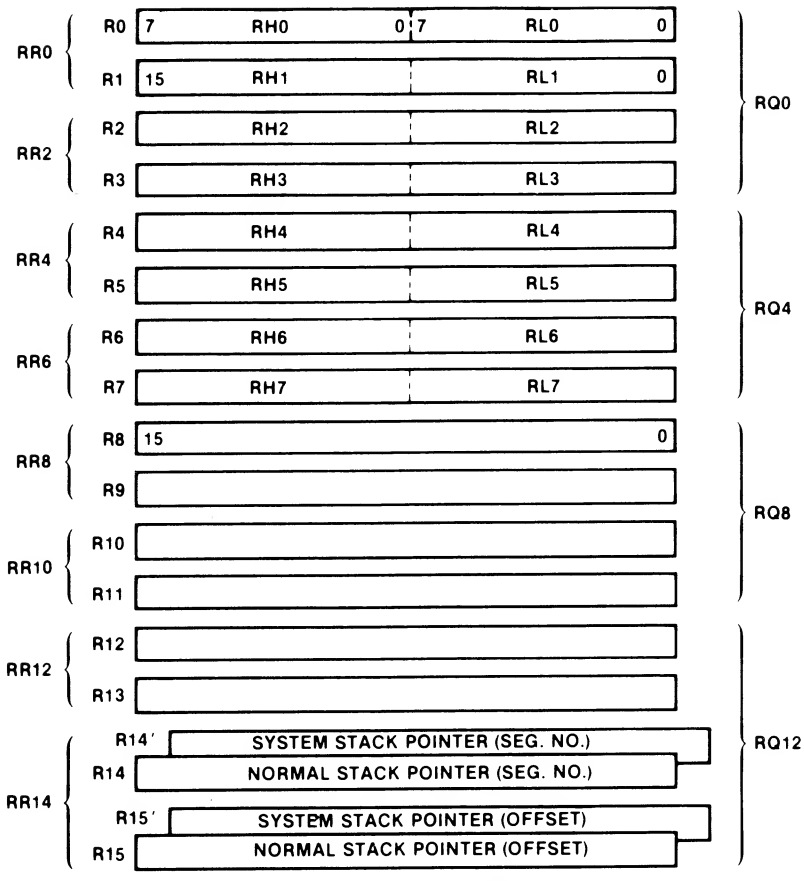


FIGURE 5.2 Zilog Z8001 general-purpose registers. Reproduced by permission. © 1979 Zilog, Inc. This material shall not be reproduced without the written consent of Zilog, Inc.

identifier is the first word of the trapped instruction. The new program status is then automatically loaded from the program status area into the system memory designated by the program status area pointer (PSAP). The program status area in memory is shown in Figure 5.8.

It should be clear that the "modes" of the Z8000 are completely distinct from the maximum and minimum modes of the Intel 8086. In the latter the word mode refers to a hardware configuration and the significance of certain status signals generated on pins 24–31. In the Z8000 the word mode is an internal software operational environment.

System Configuration

The Z8001 utilizes a segmented addressing space similar to that of the Intel 8086, while the Z8002 is a non-segmented version of the Z8001. The Z8002 is able to address memory directly, while the Z8001 requires an external memory management unit, the Z8010 for deriving the actual physical address of memory.

As we point out above, the Z8001 has an 8-megabyte address space. The address space is divided into 128 segments of 64K bytes each. These segments are addressed by means of a 7-bit segment address that points to the segment and a 16-bit offset address that specifies the location relative to the beginning of the segment. Just as in the Intel 8086, the two parts of the segmented address—the segment and the offset— may be manipulated separately by the user.

The segment:offset combination is known as the logical address. To form the physical address, or the actual memory location, the segment and offset data must be manipulated and the physical address computed. In the Intel 8086 this is done internally. In the Z8001 it is done externally in a separate component known as the Z8010 Memory Management Unit, shown in block diagram form in Figure 5.9.

As is suggested in Figure 5.9, the 7-bit segment number is applied to a base address register file that produces

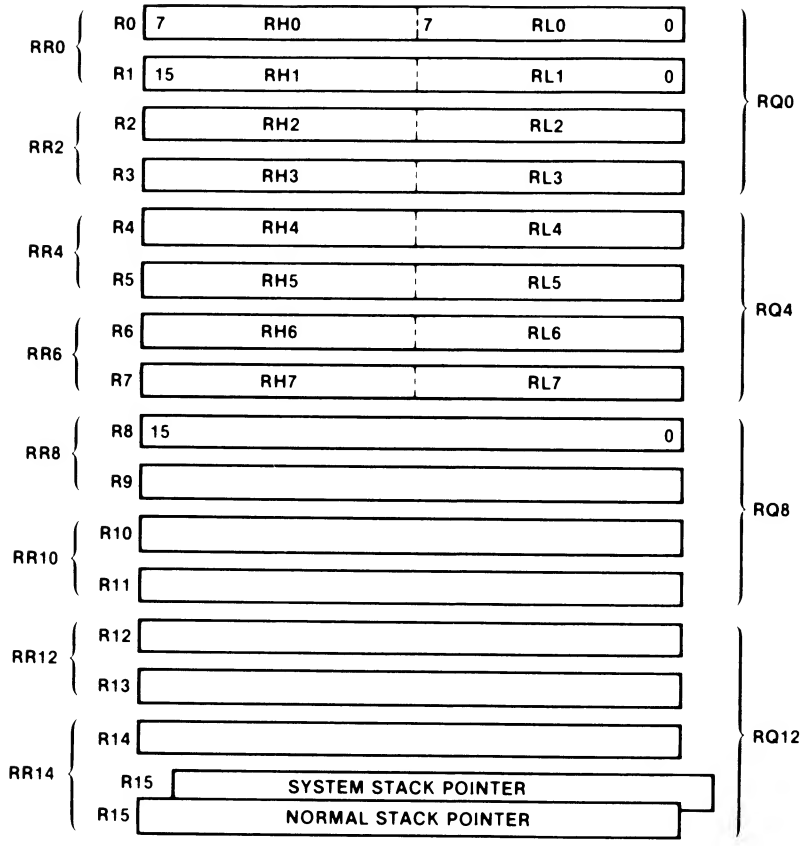


FIGURE 5.3 Zilog Z8002 general-purpose registers. Reproduced by permission. © 1979 Zilog, Inc. This material shall not be reproduced without the written consent of Zilog, Inc.

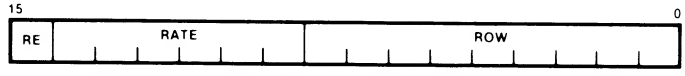


FIGURE 5.4 Zilog Z8000 refresh counter. Reproduced by permission. © 1979 Zilog, Inc. This material shall not be reproduced without the written consent of Zilog, Inc.

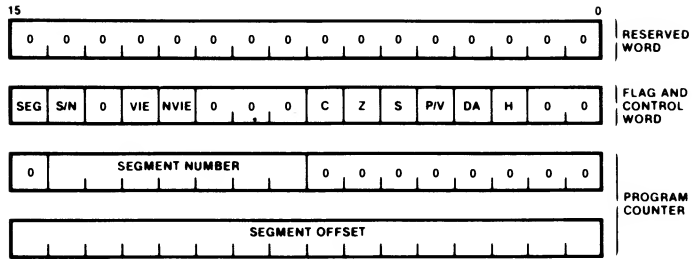
the base address of the physical memory segment. The 16-bit offset is then added thereto to obtain the actual physical address, which is 24-bits long. This must be compared with the 20-bit physical address space of the Intel 8086, thus indicating why the Z8000 has 8-megabyte address capability while the 8086 has 1-megabyte address capability.

The address translation performed in the Z8010 on the segment portion of the logical address is known as *relocation*. The segments are relocatable. Each memory management unit stores 64 entries consisting of the segment base address and its attributes, size, and status. Segments may be defined by the user to be anywhere in size from 256 to 64K bytes in increments of 256 bytes. It is even possible to use memory management units to create multiple translation tables.

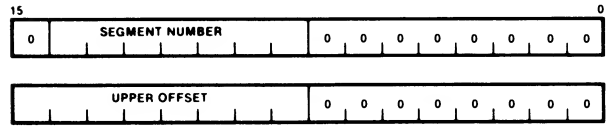
Thus there are similarities between the segmented addressing of the Z8001 and that of the Intel 8086. The base addresses in the Z8000 system are stored in the Z8010. The base addresses in the 8086 are stored in the segment registers in the 8086. But the differences between the Z8000 and the 8086 are more striking. There are four segment registers in the 8086, versus 64 in the Z8010. However, the segment registers in the 8086 may be readily changed, and there are segment override options available to the user.

Further details concerning segment addressing in the Z8000 are presented in the section on the instruction set.

It should be pointed out at this point that in contrasting and comparing the Z8000 with the Intel 8086 it must be realized that the Z8000, like the Z80 before

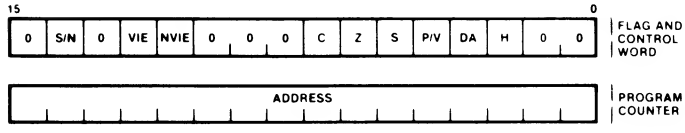

15
0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
RESERVED WORD
SEG S/N 0 VIE NVIE 0 0 0 C Z S P/V DA H 0 0
FLAG AND CONTROL WORD
0 SEGMENT NUMBER 0 0 0 0 0 0 0 0
SEGMENT OFFSET
PROGRAM COUNTER

Z8001 Program Status Registers

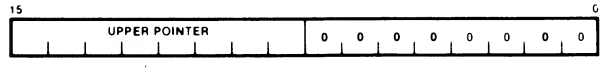


Z8001 Program Status Area Pointer

FIGURE 5.5 Zilog Z8001 program status registers. Reproduced by permission. © 1979 Zilog, Inc. This material shall not be reproduced without the written consent of Zilog, Inc.



Z8002 Program Status Registers



Z8002 Program Status Area Pointer

FIGURE 5.6 Zilog Z8002 program status register. Reproduced by permission. © 1979 Zilog, Inc. This material shall not be reproduced without the written consent of Zilog, Inc.

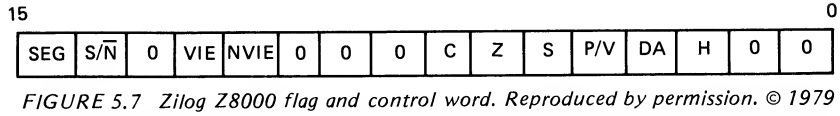


FIGURE 5.7 Zilog Z8000 flag and control word. Reproduced by permission. © 1979 Zilog, Inc. This material shall not be reproduced without the written consent of Zilog, Inc.

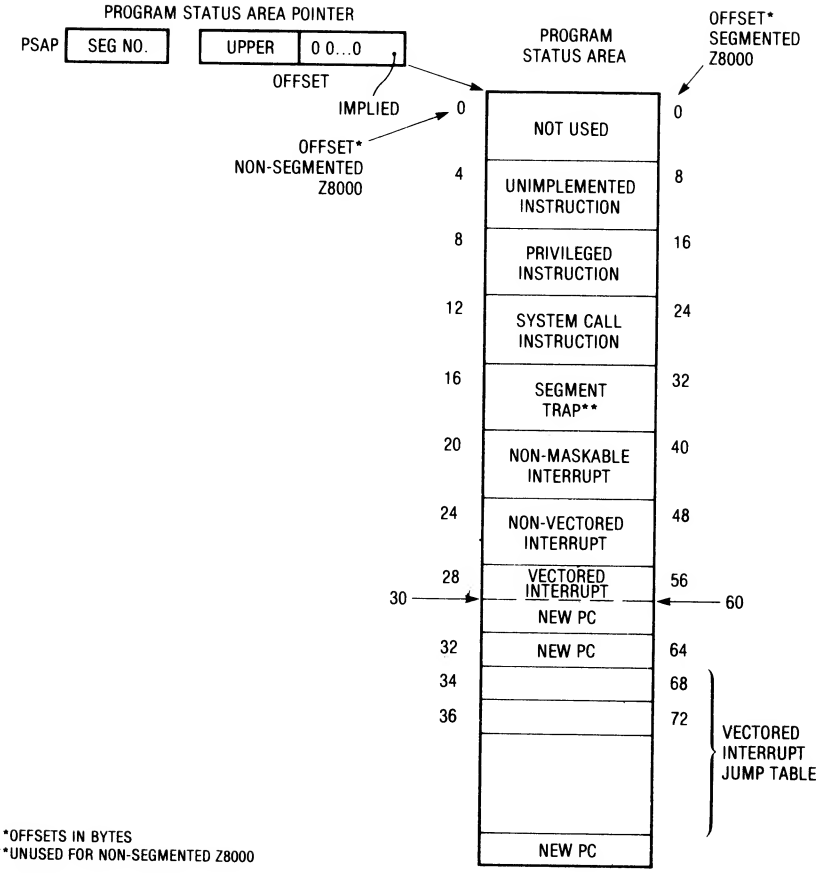


FIGURE 5.8 Z8000 program status area. Reproduced by permission. © 1979 Zilog, Inc. This material shall not be reproduced without the written consent of Zilog, Inc.

it, is a product that developed slightly later than its competitor and therefore may represent a design and performance improvement thereover.

System Timing

System timing is important for understanding the hardware capabilities and limitations of a particular microprocessor. In the present section we consider some of the basic timing diagrams of the Z8000, namely, memory read and write, and interrupt and segment trap request and acknowledge.

Memory Read and Write

The memory read and write timing diagram is shown in Figure 5.10. In the Z8001, the 7-bit segment number appears on the segment number pins SN_0-SN_6 earlier than the 16-bit address offset to permit the Memory Management Unit to perform the relocation operation. The address strobe then indicates that a valid address appears on the address pins AD_0-AD_{15}. The device reads or writes depending on the value of the $\overline{R/W}$ line, as shown in the lower half of the figure. $\overline{DS}$ is the data strobe that indicates that the information appearing on the data lines is valid.

Interrupt and Segment Trap Request/Acknowledge

The interrupt and segment trap request/acknowledge is somewhat more complicated and should be considered here. The timing of such a cycle is shown in Figure 5.11.

The figure shows both a nonmaskable interrupt ($\overline{NMI}$) and a vectored, nonvectored, or segmentation trap signal ($\overline{VI}$, $\overline{NVI}$, $\overline{SEGT}$) applied to the Z8000. Note that the asynchronous $\overline{NMI}$ signal applied externally sets an internal synchronous $\overline{NMI}$ signal. The next instruction fetch cycle is aborted, and the microprocessor enters the acknowledge cycle. There are several important points that should be noted here:

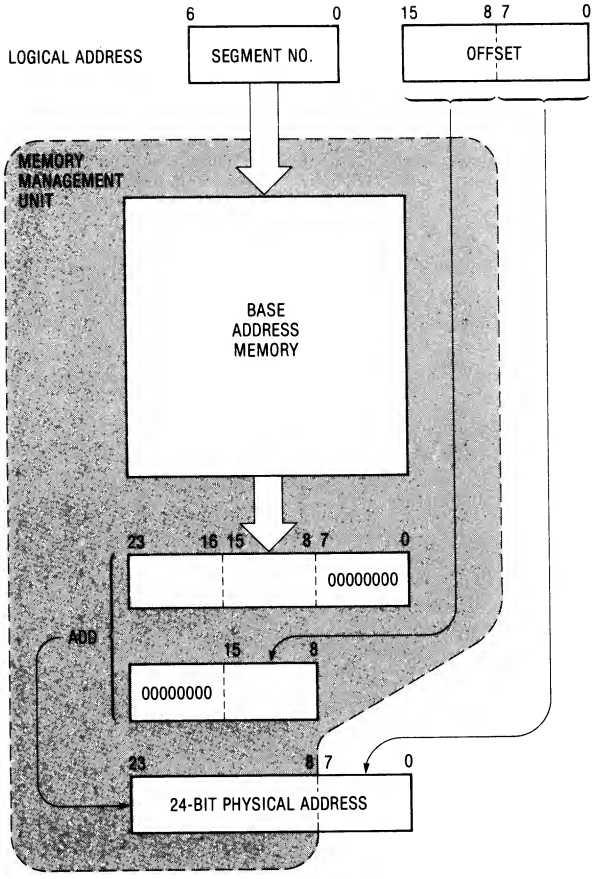


FIGURE 5.9 Zilog Z8010 logical to physical address translation. Reproduced by permission. © 1979 Zilog, Inc. This material shall not be reproduced without the written consent of Zilog, Inc.

- The microprocessor changes from normal to system mode, the N/$\overline{S}$ signal line goes low.
- Five automatic wait states are inserted in the machine cycle.
- The status lines indicate that an acknowledge cycle is taking place.
- The data strobe line ($\overline{DS}$) goes low, and an identifier appears on AD_0-AD_{15}. The word identifies the source of the interrupt or trap.

The status lines are a group of four signal lines that supply status information during the machine cycle. In Z8000 they are more associated with interrupt and segment trap acknowledgment and other specialized operations than with normal I/O operation, as they are in the Intel 8086.

The status line codes in the Z8000 are defined in Table 5.2.

Instruction Set

The instruction set of the Z8000 includes the following categories:

- Load and Exchange.
- Arithmetic.
- Logical.
- Program control.
- Bit manipulation.
- Rotate and shift.
- Block transfer and string manipulation.
- Input/output.
- CPU control.

Before we consider each of the instructions, it is first necessary to list the addressing modes with the Z8000:

- Register (RM).
- Immediate (IM).
- Indirect register (IR).
- Direct address (DA).
- Index (X).
- Relative address (RA).
- Base address (BA).
- Base index (BX).

Figure 5.12 shows the various addressing modes, indicating for each mode the method of deriving the operand address and the operand value. The addressing modes of the Z8000 are essentially similar to those of the Intel 8086, and therefore further explanation is not necessary.

In listing the instruction set of the Z8000, a brief definition of some notation used would be helpful. The Z8000 instructions operate on a wide variety of data types:

- Bits.
- BCD digits (4 bits).
- Bytes (8 bits).
- Words (16 bits).
- Long words (32 bits).
- Byte strings and word strings (up to 64K bytes).

The basic data type is a byte and all other data types are referenced using their first byte address and their length in bytes.

Some of the basic instruction formats are presented in Figure 5.13; these should be considered prior to describing the complete list of instruction formats. Here we consider three types of instructions: load (LD), add (ADD), and jump relative (JR). The mode field indicates

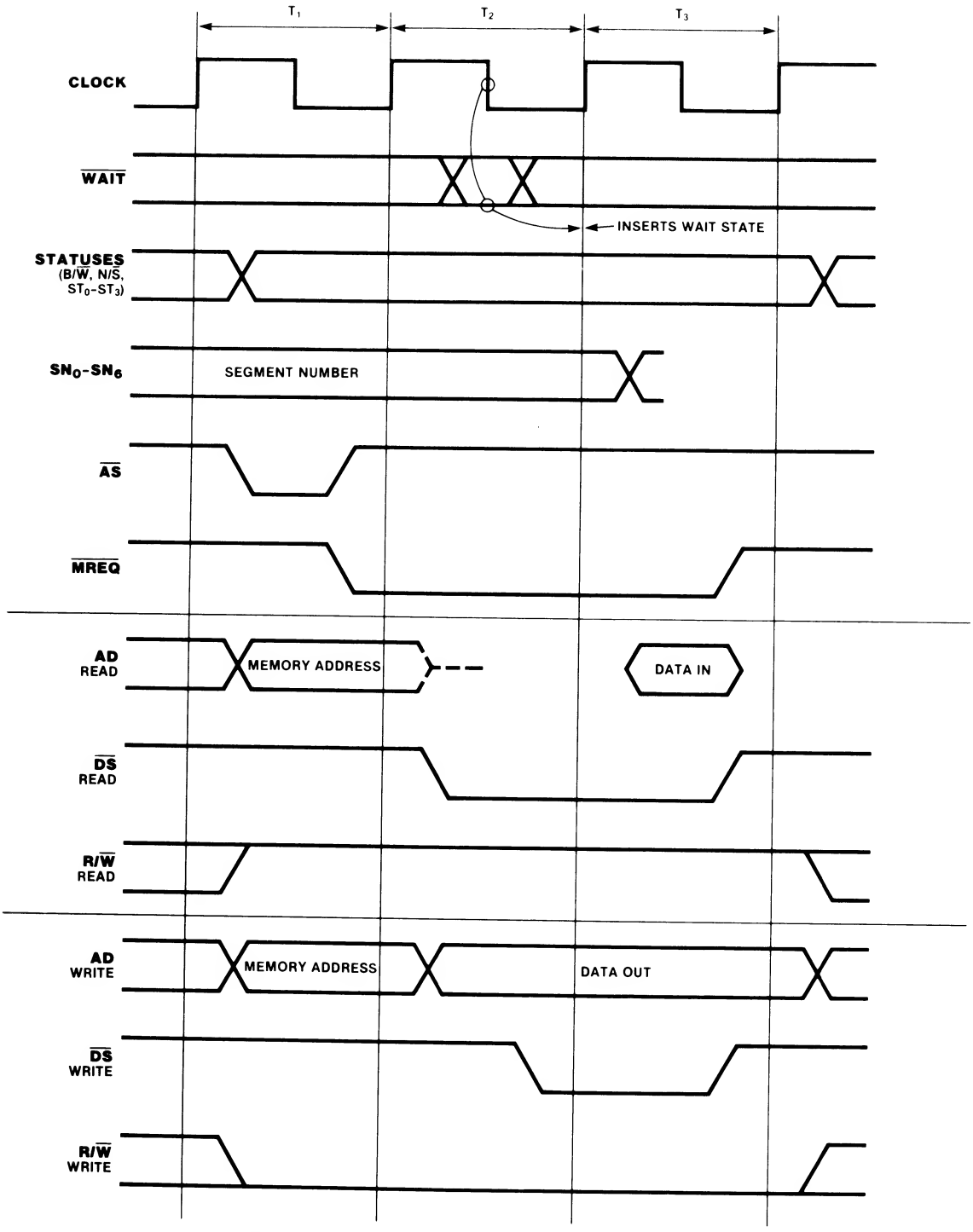


FIGURE 5.10 Zilog Z8000 Memory read and write timing diagram. Reproduced by permission. © 1979 Zilog, Inc. This material shall not be reproduced without the written consent of Zilog, Inc.

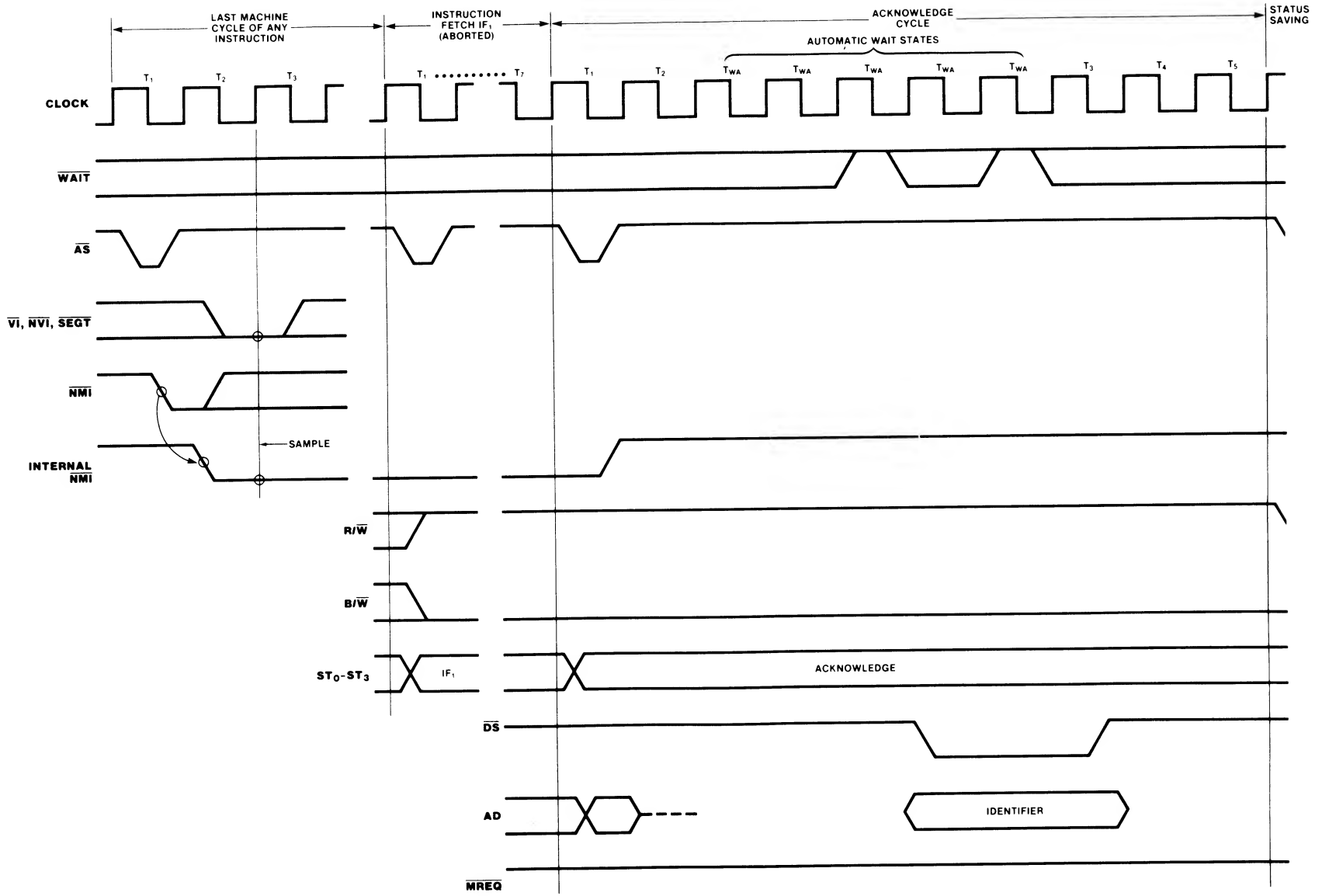


FIGURE 5.11 Zilog Z8000 interrupt and segment trap request/acknowledge timing diagram. Reproduced by permission. © 1979 Zilog, Inc. This material shall not be reproduced without the written consent of Zilog, Inc.

Mode	Operand Addressing			Operand Value
	In the Instruction	In a Register	In Memory	
Register		REGISTER ADDRESS → OPERAND		The content of the register
Immediate	OPERAND			In the instruction
Indirect Register		REGISTER ADDRESS → ADDRESS →	OPERAND	The content of the location whose address is in the register
Direct Address		ADDRESS →	OPERAND	The content of the location whose address is in the instruction
Index		REGISTER ADDRESS + BASE ADDRESS, DISPLACEMENT → + →	OPERAND	The content of the location whose address is the address in the instruction, offset by the content of the working register
Relative Address		DISPLACEMENT, PC VALUE → + →	OPERAND	The content of the location whose address is the content of the program counter, offset by the displacement in the instruction
Base Address		REGISTER ADDRESS, DISPLACEMENT → BASE ADDRESS, + →	OPERAND	The content of the location whose address is the address in the register, offset by the displacement in the instruction
Base Index		REGISTER ADDRESS, REGISTER ADDRESS → BASE ADDRESS, DISPLACEMENT, + →	OPERAND	The content of the location whose address is the address in the register, offset by the displacement in the register

FIGURE 5.12 Zilog Z8000 Addressing modes. Reproduced by permission. © 1979 Zilog, Inc. This material shall not be reproduced without the written consent of Zilog, Inc.

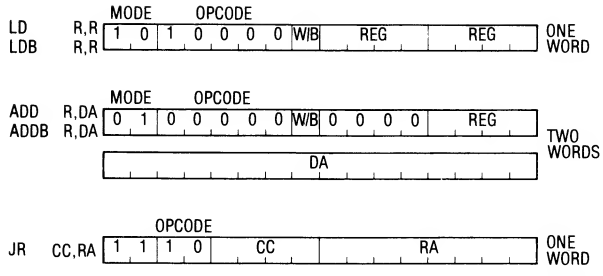


FIGURE 5.13 Zilog Z8000 Instruction format. Reproduced by permission. © 1979 Zilog, Inc. This material shall not be reproduced without the written consent of Zilog, Inc.

the addressing mode, for example, register (R) and direct address (DA). The op code indicates the specific instruction, for example, 1 0 0 0 0 for load, 0 0 0 0 0 for add. The data element type field W/B specifies whether a word or a byte is being designated, and the remaining fields are register designator fields. In the instruction listing to follow, "src" is the source operand, and "dst" is the destination operand. In some instructions, such as ADD R, DA and JR CC, RA there are additional fields for special elements. DA refers to the direct address, CC to the condition code descriptor, and RA to the relative address. The specific meaning of these special fields becomes evident below as each instruction is described in more detail.

In Table 5.3 we summarize the Z8000 instruction set, specifying the:

- Mnemonics.
- Operands.
- Addressing modes.
- Clock cycles for word/byte, and long word operation, and for segmented or nonsegmented operation.
- Operation.

Although this table is useful for studying the structure of the Z8000 instruction set, for actual programming purposes a more detailed listing, including the 1's and 0's representation, is necessary. Such a detailed listing may be found in the Z8000 CPU Instruction Set published by Zilog, Inc.

Acknowledgment

ZILOG, Z8001, and Z8002 are all trademarks of Zilog, Inc., with whom the publisher and author are not associated.

References

Z8001, Z8002 Product Specification (Preliminary, March 1979), Zilog, Inc., Cupertino, Calif., 1979.

Z8000 Instruction Set, Zilog, Inc., Cupertino, Calif., 1979.

TABLE 5.2. Status Line Code[a]

Status Line Codes	ST_3-ST_0	Definition	ST_3-ST_0	Definition
	0 0 0 0	Internal operation	1 0 0 0	Data memory request
	0 0 0 1	Memory refresh	1 0 0 1	Stack memory request
	0 0 1 0	I/O reference	1 0 1 0	Reserved
	0 0 1 1	Special I/O reference (e.g., to an MMU)	1 0 1 1	Reserved
	0 1 0 0	Segment trap acknowledge	1 1 0 0	Program reference, nth word
	0 1 0 1	Non-maskable interrupt acknowledge	1 1 0 1	Instruction fetch, first word
	0 1 1 0	Non-vectored interrupt acknowledge	1 1 1 0	Reserved
	0 1 1 1	Vectored interrupt acknowledge	1 1 1 1	Reserved

[a] *Reproduced by permission. © 1979 Zilog, Inc. This material shall not be reproduced without the written consent of Zilog, Inc.*

TABLE 5.3. Zilog Z8000 Instruction Set Summary[a]

Instruction Set Summary

The Z8000 provides the following types of instructions:

- ☐ Load and Exchange
- ☐ Arithmetic
- ☐ Logical
- ☐ Program Control
- ☐ Bit Manipulation
- ☐ Rotate and Shift
- ☐ Block Transfer and String Manipulation
- ☐ Input/Output
- ☐ CPU Control

Load and Exchange

Mnemonics	Operands	Addr. Modes	Clock Cycles* Word, Byte NS	SS	SL	Long Word NS	SS	SL	Operation
CLR **CLRB**	dst	R	7	-	-				**Clear** dst ← 0
		IR	8	-	-				
		DA	11	12	14				
		X	11	12	15				
EX **EXB**	R, src	R	6	-	-				**Exchange** R ↔ src
		IR	12	-	-				
		DA	15	16	18				
		X	16	16	19				
LD **LDB** **LDL**	R, src	R	3	-	-	5	-	-	**Load into Register** R ← src
		IM	7	-	-	11	-	-	
		IM	5 (byte only)						
		IR	7	-	-	11	-	-	
		DA	9	10	12	12	13	15	
		X	10	10	13	13	13	16	
		BA	14	-	-	17	-	-	
		BX	14	-	-	17	-	-	
LD **LDB** **LDL**	dst, R	IR	8	-	-	11	-	-	**Load into Memory** (Store) dst ← R
		DA	11	12	14	14	15	17	
		X	12	12	15	15	15	18	
		BA	14	-	-	17	-	-	
		BX	14	-	-	17	-	-	
LD **LDB**	dst, IM	IR	11	-	-				**Load Immediate into Memory** dst ← IM
		DA	14	15	17				
		X	15	15	18				

*** NS = Non-Segmented SS = Segmented Short Offset SL = Segmented Long Offset**

[a] *Reproduced by permission. © 1979 Zilog, Inc. This material shall not be reproduced without the written consent of Zilog, Inc.*

TABLE 5.3. (*Continued*)

	Mnemonics	Operands	Addr. Modes	Clock Cycles: Word, Byte NS	SS	SL	Long Word NS	SS	SL	Operation
Load and Exchange (Continued)	**LDA**	R, src	DA	12	13	15				**Load Address**
			X	13	13	16				R ← source address
			BA	15	-	-				
			BX	15	-	-				
	LDAR	R, src	RA	15	-	-				**Load Address Relative**
										R ← source address
	LDK	R, src	IM	5	-	-				**Load Constant**
										R ← n (n = 0 ... 15)
	LDM	R, src, n	IR	11	-	-				**Load Multiple**
			DA	14	15	17	+3 n			R ← src (n consecutive words)
			X	15	15	18				(n = 1 ... 16)
	LDM	dst, R, n	IR	11	-	-				**Load Multiple** (Store Multiple)
			DA	14	15	17	+3 n			dst ← R (n consecutive words)
			X	15	15	18				(n = 1 ... 16)
	LDR **LDRB** **LDRL**	R, src	RA	14	-	-	17	-	-	**Load Relative** R ← src (range -32768 ... +32767)
	LDR **LDRB** **LDRL**	dst, R	RA	14	-	-	17	-	-	**Load Relative** (Store Relative) dst ← R (range -32768 ... +32767)
	POP **POPL**	dst, IR	R	8	-	-	12	-	-	**Pop**
			IR	12	-	-	19	-	-	dst ← IR
			DA	15	16	18	22	23	25	Autoincrement contents of R
			X	16	16	19	23	23	26	
	PUSH **PUSHL**	IR, src	R	9	-	-	12	-	-	**Push**
			IM	12	-	-	-	-	-	Autodecrement contents of R
			IR	13	-	-	20	-	-	IR ← src
			DA	13	14	16	20	21	23	
			X	14	14	17	21	21	24	
Arithmetic	**ADC** **ADCB**	R, src	R	5	-	-				**Add with Carry** R ← R + src + carry
	ADD **ADDB** **ADDL**	R, src	R	4	-	-	8	-	-	**Add**
			IM	7	-	-	14	-	-	R ← R + src
			IR	7	-	-	14	-	-	
			DA	9	10	12	15	16	18	
			X	10	10	13	16	16	19	
	CP **CPB** **CPL**	R, src	R	4	-	-	8	-	-	**Compare with Register**
			IM	7	-	-	14	-	-	R - src
			IR	7	-	-	14	-	-	
			DA	9	10	12	15	16	18	
			X	10	10	13	16	16	19	
	CP **CPB**	dst, IM	IR	11	-	-				**Compare with Immediate**
			DA	14	15	17				dst - IM
			X	15	15	18				
	DAB	dst	R	5	-	-				**Decimal Adjust**
	DEC **DECB**	dst, n	R	4	-	-				**Decrement by n**
			IR	11	-	-				dst ← dst - n
			DA	13	14	16				(n = 1 ... 16)
			X	14	14	17				

TABLE 5.3. (*Continued*)

	Mnemonics	Operands	Addr. Modes	Clock Cycles: Word, Byte NS	SS	SL	Long Word NS	SS	SL	Operation
Arithmetic (Continued)	**DIV** **DIVL**	R, src	R	95	-	-	723	-	-	**Divide** (signed) Word: $R_{n+1} \leftarrow R_{n,n+1} \div$ src $R_n \leftarrow$ remainder Long Word: $R_{n+2,n+3} \leftarrow R_{n\ldots n+3} \div$ src $R_{n,n+1} \leftarrow$ remainder
			IM	95	-	-	723	-	-	
			IR	95	-	-	723	-	-	
			DA	96	97	99	724	725	727	
			X	97	97	100	725	725	728	
	EXTS **EXTSB** **EXTSL**	dst	R	11	-	-	11	-	-	**Extend Sign** Extend sign of low order half of dst through high order half of dst
	INC **INCB**	dst, n	R	4	-	-				**Increment by n** dst ← dst + n (n = 1 ... 16)
			IR	11	-	-				
			DA	13	14	16				
			X	14	14	17				
	MULT **MULTL**	R, src	R	70	-	-	282*	-	-	**Multiply** (signed) Word: $R_{n,n+1} \leftarrow R_{n+1} \cdot$ src Long Word: $R_{n\ldots n+3} \leftarrow R_{n+2,\,n+3} \cdot$ src *Plus seven cycles for each 1 in the multiplicand
			IM	70	-	-	282*	-	-	
			IR	70	-	-	282*	-	-	
			DA	71	72	74	283*	284*	286*	
			X	72	72	75	284*	284*	287*	
	NEG **NEGB**	dst	R	7	-	-				**Negate** dst ← 0 - dst
			IR	12	-	-				
			DA	15	16	18				
			X	16	16	19				
	SBC **SBCB**	R, src	R	5	-	-				**Subtract with Carry** R ← R - src - carry
	SUB **SUBB** **SUBL**	R, src	R	4	-	-	8	-	-	**Subtract** R ← R - src
			IM	7	-	-	14	-	-	
			IR	7	-	-	14	-	-	
			DA	9	10	12	15	16	18	
			X	10	10	13	16	16	19	
Logical	**AND** **ANDB**	R, src	R	4	-	-				**AND** R ← R AND src
			IM	7	-	-				
			IR	7	-	-				
			DA	9	10	12				
			X	10	10	13				
	COM **COMB**	dst	R	7	-	-				**Complement** dst ← NOT dst
			IR	12	-	-				
			DA	15	16	18				
			X	16	16	19				
	OR **ORB**	R, src	R	4	-	-				**OR** R ← R OR src
			IM	7	-	-				
			IR	7	-	-				
			DA	9	10	12				
			X	10	10	13				
	TEST **TESTB** **TESTL**	dst	R	7	-	-	13	-	-	**Test** dst OR 0
			IR	8	-	-	13	-	-	
			DA	11	12	14	16	17	19	
			X	12	12	15	17	17	20	
	TCC **TCCB**	cc, dst	R	5	-	-				**Test Condition Code** Set LSB if cc is true
	XOR **XORB**	R, src	R	4	-	-				**Exclusive OR** R ← R XOR src
			IM	7	-	-				
			IR	7	-	-				
			DA	9	10	12				
			X	10	10	13				

(continued)

TABLE 5.3. (*Continued*)

	Mnemonics	Operands	Addr. Modes	Clock Cycles: Word, Byte NS	SS	SL	Long Word NS	SS	SL	Operation
Program Control	**CALL**	dst	IR	10	-	15				**Call Subroutine** Autodecrement SP @ SP ← PC PC ← dst
			DA	12	18	20				
			X	13	18	21				
	CALR	dst	RA	10	-	15				**Call Relative** Autodecrement SP @ SP ← PC PC ← PC + dst (range -4094 to +4096)
	DJNZ **DBJNZ**	R, dst	RA	11	-	-				**Decrement and Jump if Non-Zero** R ← R - 1 If R ≠ 0: PC ← PC + dst (range -254 to 0)
	IRET*	-	-	13	-	16				**Interrupt Return** PS ← @ SP Autoincrement SP
	JP	cc, dst	IR	10	-	15	(taken)			**Jump Conditional** If cc is true: PC ← dst
			IR	7	-	7	(not taken)			
			DA	7	8	10				
			X	8	8	11				
	JR	cc, dst	RA	6	-	-				**Jump Conditional Relative** If cc is true: PC ← PC + dst (range -256 to +254)
	RET	cc		10	-	13	(taken)			**Return Conditional** If cc is true: PC ← @ SP Autoincrement SP
				7	-	7	(not taken)			
	SC	src	IM	33	-	39				**System Call** Autodecrement SP @ SP ← old PS Push instruction PS ← System Call PS
Bit Manipulation	**BIT** **BITB**	dst, b	R	4	-	-				**Test Bit Static** Z flag ← NOT dst bit specified by b
			IR	8	-	-				
			DA	10	11	13				
			X	11	11	14				
	BIT **BITB**	dst, R	R	10	-	-				**Test Bit Dynamic** Z flag ← NOT dst bit specified by contents of R
	RES **RESB**	dst, b	R	4	-	-				**Reset Bit Static** Reset dst bit specified by b
			IR	11	-	-				
			DA	13	14	16				
			X	14	14	17				
	RES **RESB**	dst, R	R	10	-	-				**Reset Bit Dynamic** Reset dst bit specified by contents R
	SET **SETB**	dst, b	R	4	-	-				**Set Bit Static** Set dst bit specified by b
			IR	11	-	-				
			DA	13	14	16				
			X	14	14	17				
	SET **SETB**	dst, R	R	10	-	-				**Set Bit Dynamic** Set dst bit specified by contents of R
	TSET **TSETB**	dst	R	7	-	-				**Test and Set** S flag ← MSB of dst dst ← all 1s
			IR	11	-	-				
			DA	14	15	17				
			X	15	15	18				

*Privileged instruction. Executed in system mode only.

TABLE 5.3. (*Continued*)

	Mnemonics	Operands	Addr. Modes	Clock Cycles Word, Byte NS	SS	SL	Long Word NS	SS	SL	Operation
Rotate and Shift	**RLDB**	R, src	R	9	-	-				**Rotate Digit Left**
	RRDB	R, src	R	9	-	-				**Rotate Digit Right**
	RL **RLB**	dst, n	R R	6 for n = 1 7 for n = 2						**Rotate Left** by n bits (n = 1, 2)
	RLC **RLCB**	dst, n	R R	6 for n = 1 7 for n = 2						**Rotate Left through Carry** by n bits (n = 1, 2)
	RR **RRB**	dst, n	R R	6 for n = 1 7 for n = 2						**Rotate Right** by n bits (n = 1, 2)
	RRC **RRCB**	dst, n	R R	6 for n = 1 7 for n = 2						**Rotate Right through Carry** by n bits (n = 1, 2)
	SDA **SDAB** **SDAL**	dst, R	R	(15 + 3 n)			(15 + 3 n)			**Shift Dynamic Arithmetic** Shift dst left or right by contents of R
	SDL **SDLB** **SDLL**	dst, R	R	(15 + 3 n)			(15 + 3 n)			**Shift Dynamic Logical** Shift dst left or right by contents of R
	SLA **SLAB** **SLAL**	dst, n	R	(13 + 3 n)			(13 +3 n)			**Shift Left Arithmetic** by n bits
	SLL **SLLB** **SLLL**	dst, n	R	(13 + 3 n)			(13 + 3 n)			**Shift Left Logical** by n bits
	SRA **SRAB** **SRAL**	dst, n	R	(13 + 3 n)			(13 + 3 n)			**Shift Right Arithmetic** by n bits
	SRL **SRLB** **SRLL**	dst, n	R	(13 + 3 n)			(13 + 3 n)			**Shift Right Logical** by n bits
Block Transfer and String Manipulation	**CPD** **CPDB**	R_X, src, R_Y, cc	IR	20	-	-				**Compare and Decrement** R_X - src Autodecrement src address $R_Y \leftarrow R_Y - 1$
	CPDR **CPDRB**	R_X, src, R_Y, cc	IR	(11 + 9 n)						**Compare, Decrement and Repeat** R_X - src Autodecrement src address $R_Y \leftarrow R_Y - 1$ Repeat until cc is true or $R_Y = 0$
	CPI **CPIB**	R_X, src, R_Y, cc	IR	20	-	-				**Compare and Increment** R_X - src Autoincrement src address $R_Y \leftarrow R_Y - 1$
	CPIR **CPIRB**	R_X, src, R_Y, cc	IR	(11 + 9 n)						**Compare, Increment and Repeat** R_X - src Autoincrement src address $R_Y \leftarrow R_Y - 1$ Repeat until cc is true or $R_Y = 0$
	CPSD **CPSDB**	dst, src, R, cc	IR	25	-	-				**Compare String and Decrement** dst - src Autodecrement dst and src addresses R ← R - 1

(continued)

TABLE 5.3. (*Continued*)

Block Transfer and String Manipulation (Continued)	Mnemonics	Operands	Addr. Modes	Clock Cycles Word, Byte NS	SS	SL	Long Word NS	SS	SL	Operation
	CPSDR **CPSDRB**	dst, src, R, cc	IR	(11 + 14 n)						**Compare String, Decr. and Repeat** dst - src Autodecrement dst and src addresses R ← R - 1 Repeat until cc is true or R = 0
	CPSI **CPSIB**	dst, src, R, cc	IR	25	-	-				**Compare String and Increment** dst - src Autoincrement dst and src addresses R ← R - 1
	CPSIR **CPSIRB**	dst, src, R, cc	IR	(11 + 14 n)						**Compare String, Incr. and Repeat** dst - src Autoincrement dst and src addresses R ← R - 1 Repeat until cc is true or R = 0
	LDD **LDDB**	dst, src, R	IR	20	-	-				**Load and Decrement** dst ← src Autodecrement dst and src addresses R ← R - 1
	LDDR **LDDRB**	dst, src, R	IR	(11 + 9 n)						**Load, Decrement and Repeat** dst ← src Autodecrement dst and src addresses R ← R - 1 Repeat until R = 0
	LDI **LDIB**	dst, src, R	IR	20	-	-				**Load and Increment** dst ← src Autoincrement dst and src addresses R ← R - 1
	LDIR **LDIRB**	dst, src, R	IR	(11 + 9 n)						**Load, Increment and Repeat** dst ← src Autoincrement dst and src addresses R ← R - 1 Repeat until R = 0
	TRDB	dst, src, R	IR	25	-	-				**Translate and Decrement** dst ← src (dst) Autodecrement dst address R ← R - 1
	TRDRB	dst, src, R	IR	(11 + 14 n)						**Translate, Decrement and Repeat** dst ← src (dst) Autodecrement dst address R ← R - 1 Repeat until R = 0
	TRIB	dst, src, R	IR	25	-	-				**Translate and Increment** dst ← src (dst) Autoincrement dst address R ← R - 1
	TRIRB	dst, src, R	IR	(11 + 14 n)						**Translate, Increment and Repeat** dst ← src (dst) Autoincrement dst address R ← R - 1 Repeat until R = 0
	TRTDB	src1, src2, R	IR	25	-	-				**Translate and Test, Decrement** RH1 ← src 2 (src 1) Autodecrement src 1 address R ← R - 1

TABLE 5.3. (*Continued*)

Block Transfer and String Manipulation (Continued)	Mnemonics	Operands	Addr. Modes	Clock Cycles Word, Byte NS	SS	SL	Long Word NS	SS	SL	Operation
	TRTDRB	src 1, src 2, R	IR	(11 + 14 n)						**Translate and Test, Decr. and Repeat** RH1 ← src 2 (src 1) Autodecrement src 1 address R ← R - 1 Repeat until R = 0 or RH1 = 0
	TRTIB	src 1, src 2, R	IR	25						**Translate and Test, Increment** RH1 ← src 2 (src 1) Autoincrement src 1 address R ← R - 1
	TRTIRB	src 1, src 2, R	IR	(11 + 14 n)						**Translate and Test, Incr. and Repeat** RH1 ← src 2 (src 1) Autoincrement src 1 address R ← R - 1 Repeat until R = 0 or RH1 = 0
Input/ Output	**IN***	R, src	IR	10	-	-				**Input** R ← src
	INB*		DA	12	-	-				
	IND* **INDB***	dst, src, R	IR	21	-	-				**Input and Decrement** dst ← src Autodecrement dst address R ← R - 1
	INDR* **INDRB***	dst, src, R	IR	(11 + 10 n)						**Input, Decrement and Repeat** dst ← src Autodecrement dst address R ← R - 1 Repeat until R = 0
	INI* **INIB***	dst, src, R	IR	21	-	-				**Input and Increment** dst ← src Autoincrement dst address R ← R - 1
	INIR* **INIRB***	dst, src, R	IR	(11 + 10 n)						**Input, Increment and Repeat** dst ← src Autoincrement dst address R ← R - 1 Repeat until R = 0
	OUT*	dst, R	IR	10	-	-				**Output** dst ← R
	OUTB*		DA	12	-	-				
	OUTD* **OUTDB***	dst, src, R	IR	21	-	-				**Output and Decrement** dst ← src Autodecrement src address R ← R - 1
	OTDR* **OTDRB***	dst, src, R	IR	(11 + 10 n)						**Output, Decrement and Repeat** dst ← src Autodecrement src address R ← R - 1 Repeat until R = 0
	OUTI* **OUTIB***	dst, src, R	IR	21	-	-				**Output and Increment** dst ← src Autoincrement src address R ← R - 1
	OTIR* **OTIRB***	dst, src, R	IR	(11 + 10 n)						**Output, Increment and Repeat** dst ← src Autoincrement src address R ← R - 1 Repeat until R = 0

*Privileged instructions. Executed in system mode only.

(continued)

TABLE 5.3. (*Continued*)

	Mnemonics	Operands	Addr. Modes	Clock Cycles Word, Byte NS	SS	SL	Long Word NS	SS	SL	Operation
Input/Output (Continued)	**SIN*** **SINB***	R, src	DA	12	-	-				**Special Input** R ← src
	SIND* **SINDB***	dst, src, R	IR	21	-	-				**Special Input and Decrement** dst ← src Autodecrement dst address R ← R - 1
	SINDR* **SINDRB***	dst, src, R	IR	(11 + 10 n)						**Special Input, Decrement and Repeat** dst ← src Autodecrement dst address R ← R - 1 Repeat until R = 0
	SINI* **SINIB***	dst, src, R	IR	21	-	-				**Special Input and Increment** dst ← src Autoincrement dst address R ← R - 1
	SINIR* **SINIRB***	dst, src, R	IR	(11 + 10 n)						**Special Input, Increment and Repeat** dst ← src Autoincrement dst address R ← R - 1 Repeat until R = 0
	SOUT* **SOUTB***	dst, src	DA	12	-	-				**Special Output** dst ← src
	SOUTD* **SOUTDB***	dst, src, R	IR	21	-	-				**Special Output and Decrement** dst ← src Autodecrement src address R ← R - 1
	SOTDR* **SOTDRB***	dst, src, R	IR	(11 + 10 n)						**Special Output, Decr. and Repeat** dst ← src Autodecrement src address R ← R - 1 Repeat until R = 0
	SOUTI* **SOUTIB***	dst, src, R	IR	21	-	-				**Special Output and Increment** dst ← src Autoincrement src address R ← R - 1
	SOTIR* **SOTIRB***	dst, src, R	R	(11 + 10 n)						**Special Output, Incr. and Repeat** dst ← src Autoincrement src address R ← R - 1 Repeat until R = 0
CPU Control	**COMFLG**	flags	-	7	-	-				**Complement Flag** (Any combination of C, Z, S, P/V)
	DI*	int	-	6	-	-				**Disable Interrupt** (Any combination of NVI, VI)
	EI*	int	-	6	-	-				**Enable Interrupt** (Any combination of NVI, VI)
	HALT*	-	-	(8 + 3 n)						**HALT**
	LDCTL*	CTLR, src	R	7	-	-				**Load into Control Register** CTLR ← src
	LDCTL*	dst, CTLR	R	7	-	-				**Load from Control Register** dst ← CTLR

*Privileged instructions. Executed in system mode only.

TABLE 5.3. (*Continued*)

CPU Control (Continued)

Mnemonics	Operands	Addr. Modes	Clock Cycles Word, Byte NS	SS	SL	Long Word NS	SS	SL	Operation
LDCTLB	FLGR, src	R	7	-	-				**Load into Flag Byte Register** FLGR ← src
LDCTLB	dst, FLGR	R	7	-	-				**Load from Flag Byte Register** dst ← FLGR
LDPS*	src	IR	12	-	16				**Load Program Status** PS ← src
		DA	16	20	22				
		X	17	20	23				
MBIT*	-	-	7	-	-				**Test Multi-Micro Bit** Set S if $\overline{M_I}$ is Low; reset S if $\overline{M_I}$ is High.
MREQ*	dst	R	(12 + 7 n)						**Multi-Micro Request**
MRES*	-	-	5	-	-				**Multi-Micro Reset**
MSET*	-	-	5	-	-				**Multi-Micro Set**
NOP	-	-	7	-	-				**No Operation**
RESFLG	flag	-	7	-	-				**Reset Flag** (Any combination of C, Z, S, P/V)
SETFLG	flag	-	7	-	-				**Set Flag** (Any combination of C, Z, S, P/V)

*Privileged instructions. Executed in system mode only.

Condition Codes

Code	Meaning	Flag Settings	CC Field
	Always false	-	0000
	Always true	-	1000
Z	Zero	Z = 1	0110
NZ	Not zero	Z = 0	1110
C	Carry	C = 1	0111
NC	No Carry	C = 0	1111
PL	Plus	S = 0	1101
MI	Minus	S = 1	0101
NE	Not equal	Z = 0	1110
EQ	Equal	Z = 1	0110
OV	Overflow	P/V = 1	0100
NOV	No overflow	P/V = 0	1100
PE	Parity is even	P/V = 1	0100
PO	Parity is odd	P/V = 0	1100
GE	Greater than or equal (signed)	(S XOR P/V) = 0	1001
LT	Less than (signed)	(S XOR P/V) = 1	0001
GT	Greater than (signed)	[Z OR (S XOR P/V)] = 0	1010
LE	Less than or equal (signed)	[Z OR (S XOR P/V)] = 1	0010
UGE	Unsigned greater than or equal	C = 0	1111
ULT	Unsigned less than	C = 1	0111
UGT	Unsigned greater than	[(C = 0) AND (Z = 0)] = 1	1011
ULE	Unsigned less than or equal	(C OR Z) = 1	0011

Note that some condition codes have identical flag settings and binary fields in the instruction:
Z = EQ, NZ = NE, C = ULT, NC = UGE, OV = PE, NOV = PO

6

bit-slice microprocessors

The "bit-slice" microprocessors are intended to be connected in parallel or cascaded together to form a central processing unit of a computer with a microprogrammed architecture. These microprocessors are configured as 2-bit or 4-bit "slices," which when connected together form a computer with $2n$-bit or $4n$-bit word length, where n is the number of such microprocessors connected in parallel.

After a brief introduction to microprogramming, we consider the following bit-slice microprocessors in this chapter:

- AMD 2901/2903.
- Motorola 10800.
- Texas Instruments SN 54S/74S-481.
- Fairchild F100220.

Microprogramming

Microprogramming is a computer design technique in which control functions are performed by read-only memory (ROM) rather than random logic (arrays of gates and flip-flops). Although microprogramming was originally proposed by Dr. M. V. Wilkes in 1951, it was not until the development of suitable read-only storage techniques that microprogramming was used commercially. The IBM System/360 was one of the first computers to use microprogramming, and numerous minicomputers today are user microprogrammable.

Microprogrammed computers are more highly organized than nonmicroprogrammable systems, and they therefore have improved flexibility, performance, and LSI circuit utilization. Some of the other key advantages are:

- Diagnostics can be implemented easily.
- Design changes are relatively easy.
- Field updates are simple.
- System definition can be expanded more easily.
- Cost and design time are reduced.

A microprogrammed computer makes use of a se-

quence of microinstructions that is used to execute various machine functions. The microinstruction differs from ordinary machine instruction in that in addition to defining the operations to be carried out, it defines that address of the next microinstruction to be executed. Microinstructions are stored in a read-only store called the microprogram memory. The microprogrammed computer therefore consists of three key parts:

- The microprogram memory.
- Means for determining the next microinstruction (a sequencer).
- The ALU, which acts in response to the microinstruction.

More details on the specific nature of these components are presented as we consider the various bit-slice microprocessors.

AMD 2901/2901A

The AMD 2901A is a 4-bit bipolar microprocessor slice implemented in a 40-pin package. The functional block diagram of the 2901A is shown in Figure 6.1.

The functional block diagram indicates a microinstruction decode unit, a 16-word by 4-bit 2-port RAM, a high-speed ALU, and various shifting, decoding, and multiplexing circuitry. The microinstruction word is seen to be 9 bits in length and selects the ALU source operands, the ALU function, and the ALU destination register.

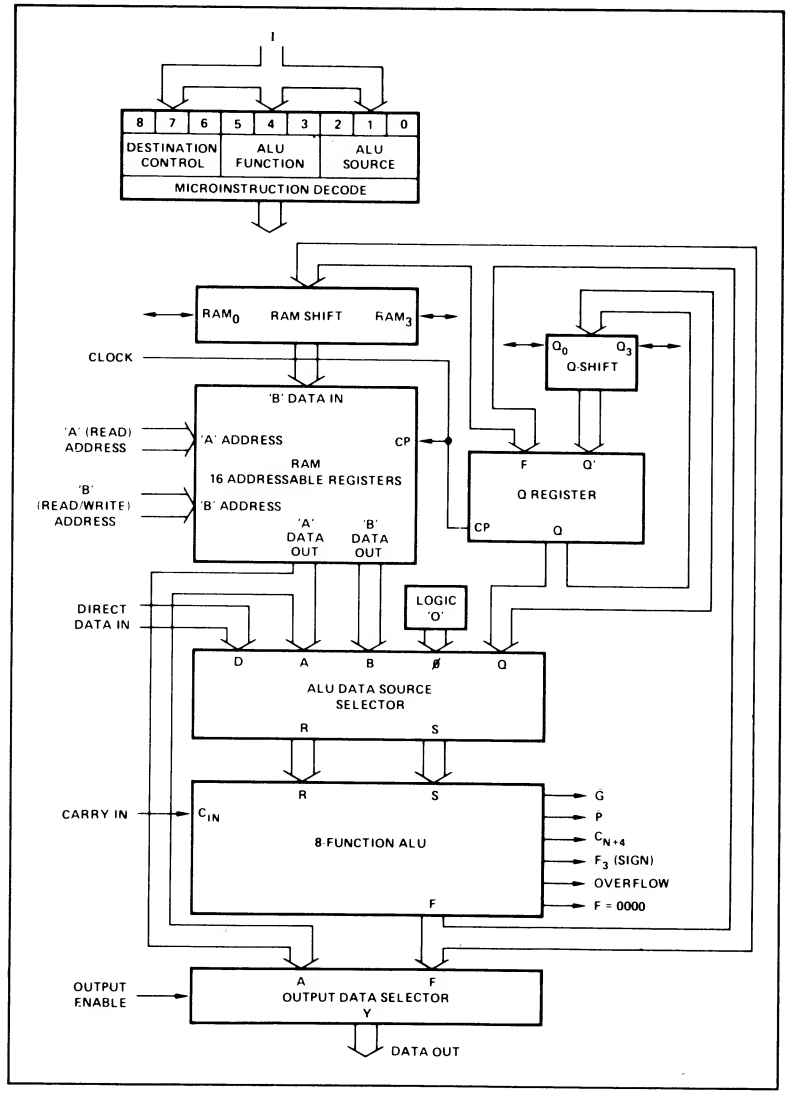


FIGURE 6.1 AMD 2901 functional block diagram. (Copyright © 1979 Advanced Micro Devices, Inc. Reproduced with permission of copyright owner.)

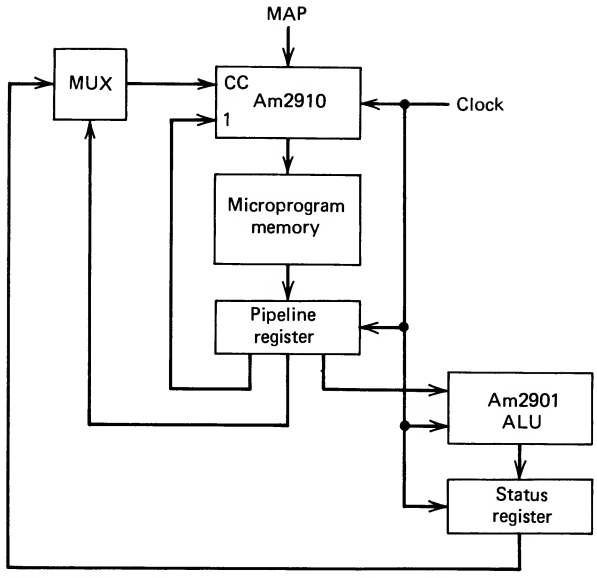


FIGURE 6.2 Typical microprogrammed system with 2901. One-level pipeline based system. (Copyright © 1979 Advanced Micro Devices, Inc. Reproduced with permission of copyright owner.)

As we point out above, the 2901A is designed to be used in a microprogrammed system. A typical example of such a system is shown in Figure 6.2. The inputs to the CPU are seen to be separate registers. The register inputs come from a ROM or PROM that acts as the microprogram memory. The address lines of the microprogram memory are driven by the microprogram sequencer, shown in Figure 6.2 to be AMD 2909s. The register between the microprogram memory and the CPU is sometimes referred to as a pipeline register since it permits overlapping of execution of an instruction by the CPU and fetching the next microinstruction from the microprogram memory.

It is important to illustrate how a plurality of 2901s can be connected to form a CPU of predetermined word length. Figure 6.3 shows how three 2903s may be connected in parallel to form a 16-bit CPU. The corresponding A, B, and I pins on all devices are connected. The carry output of the low-order device is connected to the carry input of the higher-order device, so as to form a ripple carry. Other carry configurations, such as carry lookahead, can also be implemented.

The AMD 2901 (or its successor the 2901A) is used in a wide variety of minicomputers as the central processing unit, including the Data General Nova 4 and the National Semiconductor System/400.

We return now to discussing the 2901. Figure 6.4 is a more detailed functional block diagram of the chip. The 2901 ALU is an eight function ALU—it can perform three binary arithmetic and five logic operations on the two 4-bit input words R and S. The ALU functions are:

R + S	R OR S
R - S	R AND S
S - R	R EXOR S
	R EXNOR S
	NOT R AND A

The R input field is derived from a bank of four 2-input multiplexers, while the S input field is derived from a bank of four 3-input multiplexers. The R input

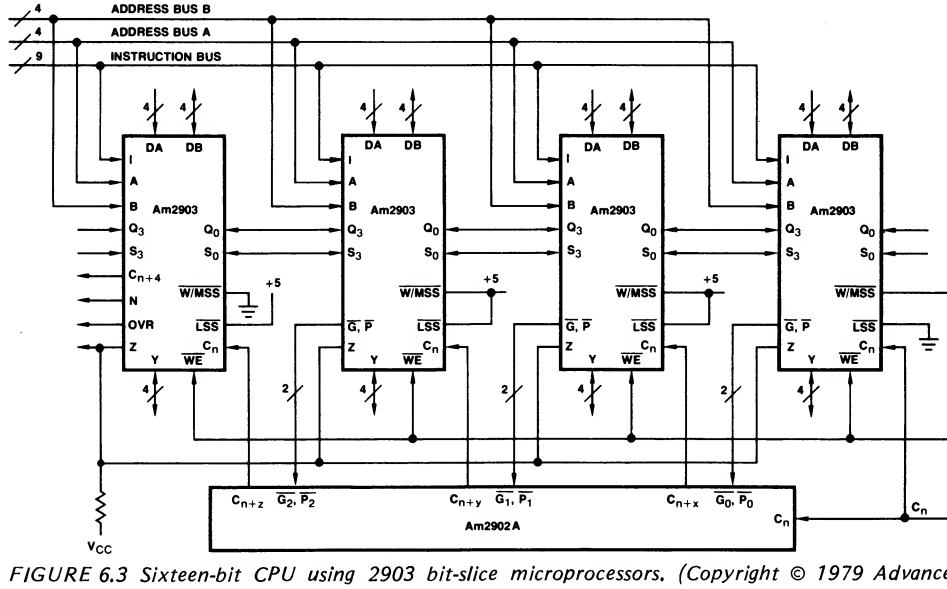


FIGURE 6.3 Sixteen-bit CPU using 2903 bit-slice microprocessors. (Copyright © 1979 Advanced Micro Devices, Inc. Reproduced with permission of copyright owner.)

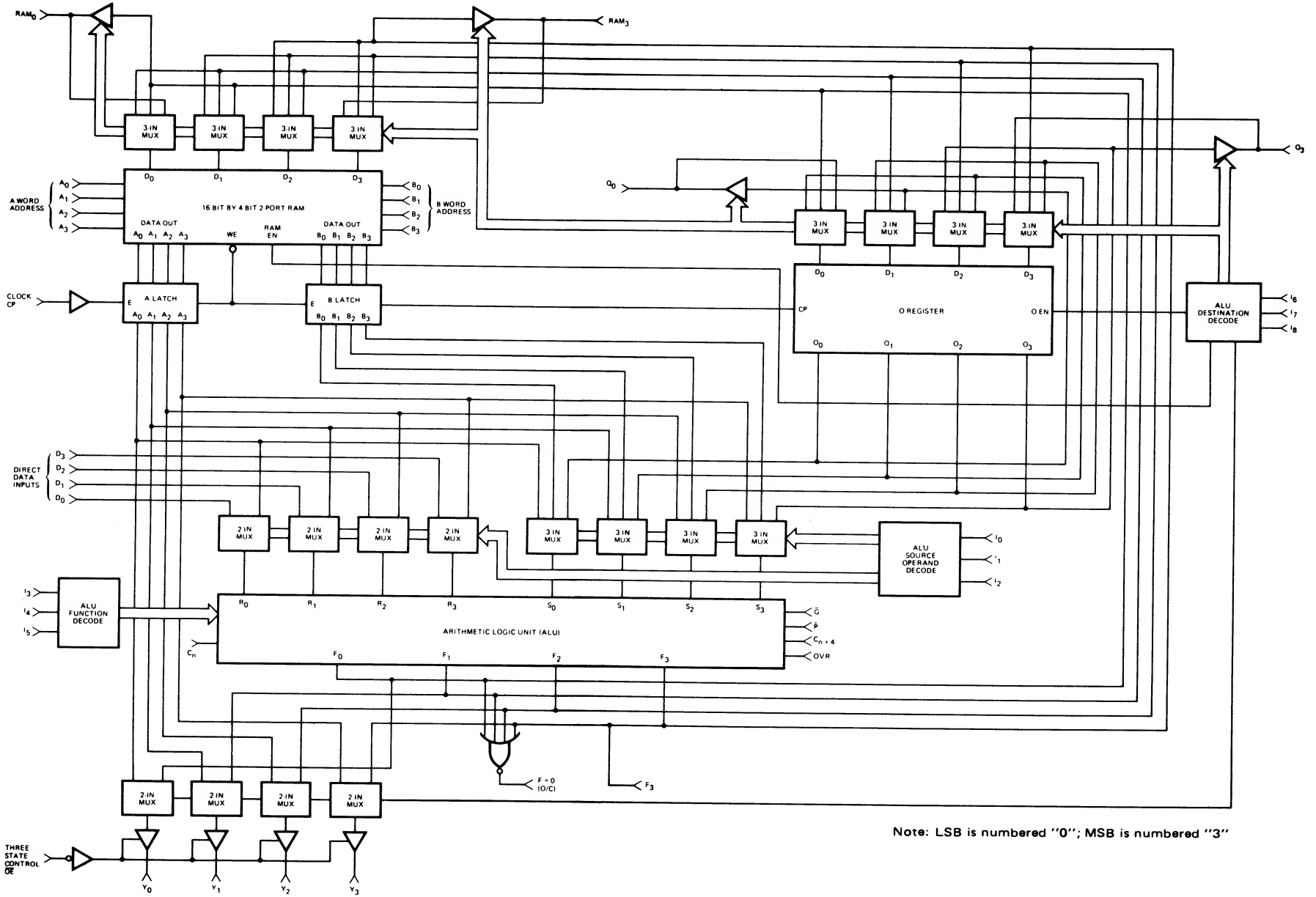


FIGURE 6.4 AMD 2901 detailed functional block diagram. (Copyright © 1979 Advanced Micro Devices, Inc. Reproduced with permission of copyright owner.)

multiplexer has the RAM A port and the direct data input D as its inputs; the S input multiplexer has the RAM A port, the RAM B port, and the Q register connected as inputs. The multiplexers function as an ALU data source selector, as represented in the block diagram in Figure 6.1. The multipliers therefore select pairs from the possible data sources A, B, D, Q, and 0 as source operands for the ALU. The possible combinations are:

A B
A D
A Q
A 0
B D
B Q
B 0
D Q
Q 0
D 0

Since the specification of an "A" address or a "B" address is arbitrary, A D, A Q, and A 0 are really equivalent to B D, B Q, and B 0. The 2901 therefore only implements 8 of the possible 10 combinations, thus permitting the specification of the ALU source operand to be done with three bits (I_0, I_1, and I_2). The source operands implemented on the 2901 together with the microcode for I_0, I_1 and I_2 are given in Table 6.1.

As we point out above, the ALU has eight possible functions. These functions are specified by the I_3, I_4, and I_5 bits of the microcode fields according to Table 6.2.

Next we turn to the outputs of the 2901. There are various outputs associated with the carry function: $\overline{G}$, a carry generate, and $\overline{P}$, a carry propagate, are associated with a carry-lookahead architecture that can be constructed with such peripheral chips as the 2902. C_{n+4} is the normal carry output.

There are also a number of status outputs: F_3, F = 0, and OVERFLOW. F_3 is the most significant bit of the ALU and therefore is the sign bit. The F = 0 output, as the name implies, is a zero-detect; when all F outputs of the ALU are 0, F = 0 is 1. Finally, the OVERFLOW output is used to indicate when the calculation performed exceeds the available 2s complemet number range in the chip.

From the block diagram of Figure 6.4, it is noted that the F output of the ALU has a number of possible destinations. These include the Output Data Selector, the Q register, and RAM. The destination function is defined by the value of the I_6, I_7 and I_8 bits of the microinstruction. The various possibilities are given in Table 6.3.

Table 6.3 shows which registers are loaded with the ALU output in the LOAD column: the RAM, the Q register, both the RAM and the Q register, or none (i.e., neither the RAM nor the Q register).

Since the inputs to both the RAM and the Q register are driven by a multiplexer, it is possible that that ALU output to be entered therein may be shifted up one position, shifted down one position, or not shifted at all (i.e., "none" in the Table 6.3 under the SHIFT column). We should also note that shifting up is the same as F being multiplied by 2, and shifting down is the same as F being divided by 2.

Finally, there are two possibilities for the Y output from the 2901: the F data from the ALU output, or the A data output from the RAM. Clearly the F data output is more important, but the ALU destination possibilities noted in Table 6.3 do make provision for an A output in one case.

TABLE 6.1. AMD 2901 Source Operands Microcode

			ALU Source Operands	
I_2	I_1	I_0	*R*	*S*
0	0	0	A	Q
0	0	1	A	B
0	1	0	0	Q
0	1	1	0	B
1	0	0	0	A
1	0	1	D	A
1	1	0	D	Q
1	1	1	D	O

TABLE 6.2. AMD 2901 ALU Function Control

I_5	I_4	I_3	*ALU Function*
0	0	0	R + S
0	0	1	S - R
0	1	0	R - S
0	1	1	R OR S
1	0	0	R AND S
1	0	1	$\overline{R}$ AND S
1	1	0	R EX-OR S
1	1	1	R EX-NOR S

TABLE 6.3. ALU Destination Possibilities

Shift	*Load*	*Y-out*
UP	RAM	F
UP	RAM + Q	F
DOWN	RAM	F
DOWN	RAM + Q	F
NONE	NONE	F
NONE	Q	F
NONE	RAM	F
NONE	RAM	A

TABLE 6.4. AMD 2901 ALU Destination Control[a]

I_8	I_7	I_6	I_5	Hex Code	ALU Shifter Function	SIO_3		Y_3		Y_2		Y_1	Y_0	SIO_0	$\overline{\text{Write}}$	Q Reg & Shifter Function	QIO_3	QIO_0
						Most Sig. Slice	Other Slices	Most Sig. Slice	Other Slices	Most Sig. Slice	Other Slices							
L	L	L	L	0	Arith. F/2→Y	Input	Input	F_3	SIO_3	SIO_3	F_3	F_2	F_1	F_0	L	Hold	Hi-Z	Hi-Z
L	L	L	H	1	Log. F/2→Y	Input	Input	SIO_3	SIO_3	F_3	F_3	F_2	F_1	F_0	L	Hold	Hi-Z	Hi-Z
L	L	H	L	2	Arith. F/2→Y	Input	Input	F_3	SIO_3	SIO_3	F_3	F_2	F_1	F_0	L	Log. Q/2→Q	Input	Q_0
L	L	H	H	3	Log. F/2→Y	Input	Input	SIO_3	SIO_3	F_3	F_3	F_2	F_1	F_0	L	Log. Q/2→Q	Input	Q_0
L	H	L	L	4	F→Y	Input	Input	F_3	F_3	F_2	F_2	F_1	F_0	Parity	L	Hold	Hi-Z	Hi-Z
L	H	L	H	5	F→Y	Input	Input	F_3	F_3	F_2	F_2	F_1	F_0	Parity	H	Log. Q/2→Q	Input	Q_0
L	H	H	L	6	F→Y	Input	Input	F_3	F_3	F_2	F_2	F_1	F_0	Parity	H	F→Q	Hi-Z	Hi-Z
L	H	H	H	7	F→Y	Input	Input	F_3	F_3	F_2	F_2	F_1	F_0	Parity	L	F→Q	Hi-Z	Hi-Z
H	L	L	L	8	Arith. 2F→Y	F_2	F_3	F_3	F_2	F_1	F_1	F_0	SIO_0	Input	L	Hold	Hi-Z	Hi-Z
H	L	L	H	9	Log. 2F→Y	F_3	F_3	F_2	F_2	F_1	F_1	F_0	SIO_0	Input	L	Hold	Hi-Z	Hi-Z
H	L	H	L	A	Arith. 2F→Y	F_2	F_3	F_3	F_2	F_1	F_1	F_0	SIO_0	Input	L	Log. 2Q→Q	Q_3	Input
H	L	H	H	B	Log. 2F→Y	F_3	F_3	F_2	F_2	F_1	F_1	F_0	SIO_0	Input	L	Log. 2Q→Q	Q_3	Input
H	H	L	L	C	F→Y	F_3	F_3	F_3	F_3	F_2	F_2	F_1	F_0	Hi-Z	H	Hold	Hi-Z	Hi-Z
H	H	L	H	D	F→Y	F_3	F_3	F_3	F_3	F_2	F_2	F_1	F_0	Hi-Z	H	Log. 2Q→Q	Q_3	Input
H	H	H	L	E	SIO_0→Y_0, Y_1, Y_2, Y_3	SIO_0	SIO_0	SIO_0	SIO_0	SIO_0	SIO_0	SIO_0	SIO_0	Input	L	Hold	Hi-Z	Hi-Z
H	H	H	H	F	F→Y	F_3	F_3	F_3	F_3	F_2	F_2	F_1	F_0	Hi-Z	L	Hold	Hi-Z	Hi-Z

Parity = $F_3 \forall F_2 \forall F_1 \forall F_0 \forall SIO_3$ L = LOW Hi-Z = High Impedance

$\forall$ = Exclusive OR H = HIGH

[a]Copyright © 1979 Advanced Micro Devices, Inc. Reproduced with permission of copyright owner.

The ALU destination possibilities are matched with the microcode for performing such operations in Table 6.4. The first instruction listed (0 0 0) corresponds to

SHIFT	LOAD	Y-OUT
NONE	Q	F

in Table 6.3 above. Note that the value loaded into the appropriate register, when shifted, is explicitly written as F/2, 2F, Q/2, or 2Q. Note also that certain inputs labeled RAM_0, RAM_3, Q_0, and Q_3 are enabled by appropriate shift instructions.

Further details concerning system configurations with the 2901 are given in Chapter 7.

AMD 2903

The AMD 2903 is a 4-bit bipolar microprocessor slice that performs all functions of the 2901 together with additional functions particularly useful in arithmetic oriented microprogrammed systems. The 2903 was introduced in 1978 and, unlike the 2901, is packed in a 48-pin package. The functional block diagram of the 2903 is shown in Fig. 6.5.

The key feature of the 2903 is that it contains 16 internal working registers that can be expanded using the external 29705 register stack. The 2903 also implements a special set of instructions that permit multiplication, division, and other arithmetic operations to take place much more efficiently.

Since the 9-bit microinstruction word (I_8, I_7, . . . I_0) has already been introduced for the 2901, we can proceed directly with listing the instruction set for the 2903.

Table 6.5 indicates the ALU functions of the 2903. We note first that when I_4, . . . I_0 = 0 0 0 0 0, special functions are executed. These special functions are defined in Table 6.6. A quick examination of Table 6.5 reveals some notable differences from the ALU function control of the 2901: the carry input signal C_n is used as one of the variables, and significantly more arithmetic and logical functions are implemented.

The source operand selection has not been forgotten: as shown in the block diagram of Figure 6.5, the I_0 input controls the multiplexer at the S input of the ALU. The combination of the OE_B and I_0 inputs function to select either the RAM output port B, the external data input DB, or the Q register. At the R input of the ALU, the E_A input selects either the RAM output port A or the external data input DA. Moreover, during some operations zeroes can be forced to the inputs of the ALU, so a 0 input is possible as well. Comparing such source operand possibilities with that of the 2901 in Table 6.1 on page 000, we see that the 2903 offers even greater designer flexibility.

TABLE 6.5. AMD 2903 ALU Functions[a]

$\overline{E_A}$	I_0	$\overline{OE_B}$	*ALU Operand R*	*ALU Operand S*
L	L	L	RAM output A	RAM output B
L	L	H	RAM output A	$DB_{0\text{-}3}$
L	H	X	RAM output A	Q register
H	L	L	$DA_{0\text{-}3}$	RAM output B
H	L	H	$DA_{0\text{-}3}$	$DB_{0\text{-}3}$
H	H	X	$DA_{0\text{-}3}$	Q register

[a] *L = low, H = high, X = don't care*

I_4	I_3	I_2	I_1	Hex Code	ALU Functions
L	L	L	L	0	I_0 = L: Special Functions; I_0 = H: F_i = HIGH
L	L	L	H	1	F = S Minus R Minus 1 Plus C_n
L	L	H	L	2	F = R Minus S Minus 1 Plus C_n
L	L	H	H	3	F = R Plus S Plus C_n
L	H	L	L	4	F = S Plus C_n
L	H	L	H	5	F = $\overline{S}$ Plus C_n
L	H	H	L	6	F = R Plus C_n
L	H	H	H	7	F = $\overline{R}$ Plus C_n
H	L	L	L	8	F_i = LOW
H	L	L	H	9	F_i = $\overline{R_i}$ AND S_i
H	L	H	L	A	F_i = R_i EXCLUSIVE NOR S_i
H	L	H	H	B	F_i = R_i EXCLUSIVE OR S_i
H	H	L	L	C	F_i = R_i AND S_i
H	H	L	H	D	F_i = R_i NOR S_i
H	H	H	L	E	F_i = R_i NAND S_i
H	H	H	H	F	F_i = R_i OR S_i

L = LOW H = HIGH i = 0 to 3

Returning now to the definition of the instruction set, Table 6.6 gives the functions executed for I_4, . . . I_0 = 0 for various values of I_8, . . . I_5.

The destination control function in 2903 is similar to that in the 2901. As is suggested in Table 6.6, there are different input and output possibilities. SIO_0 and SIO_3 are bidirectional serial shift input/outputs and are shown in the block diagram of Figure 6.5. QIO_0 and QIO_3 are similar bidirectional serial shift input/outputs. Double length shifting is possible by connecting QIO_0 of the most significant slice to SIO_0 of the least significant slice and executing an instruction that shifts both the ALU output and the Q register.

Another important feature of the 2903 is that both arithmetic and logical shift operations are possible. An

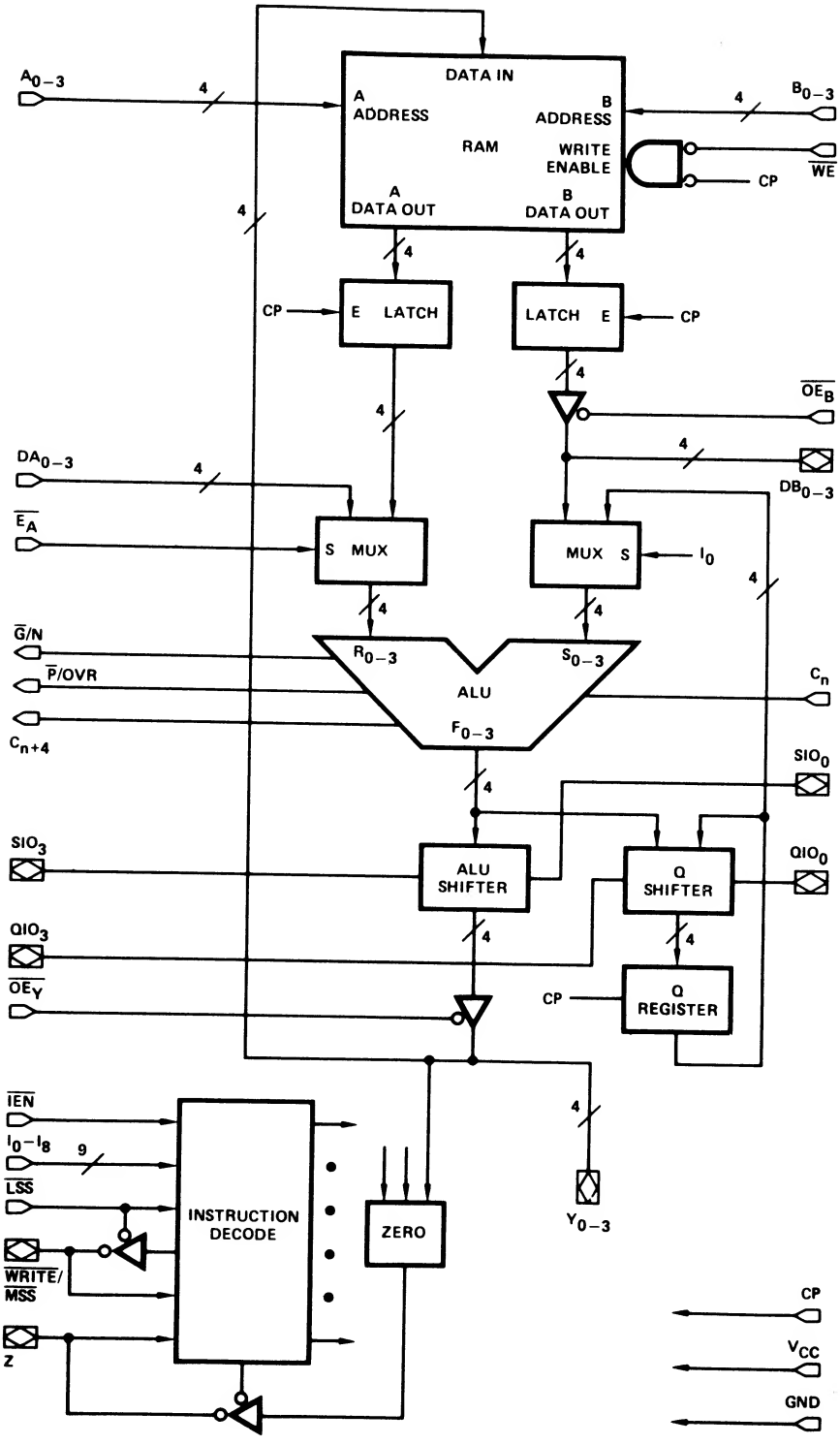


FIGURE 6.5 AMD 2903 functional block diagram. (Copyright © 1979 Advanced Micro Devices, Inc. Reproduced with permission of copyright owner.)

TABLE 6.6. AMD 2903 Special Functions[a]

I_8 I_7 I_6 I_5	Hex Code	Special Function	ALU Function	ALU Shifter Function	SIO_3 Most Sig. Slice	SIO_3 Other Slices	SIO_0	Q Reg & Shifter Function	QIO_3	QIO_0	$\overline{WRITE}$
L L L L	0	Unsigned Multiply	F= S+C_n if Z=L F=R+S+C_n if Z=H	Log. F/2→Y (Note 1)	Hi-Z	Input	F_0	Log. Q/2→Q	Input	Q_0	L
L L H L	2	Two's Complement Multiply	F=S+C_n if Z=L F=R+S+C_n if Z=H	Log. F/2→Y (Note 2)	Hi-Z	Input	F_0	Log. Q/2→Q	Input	Q_0	L
L H L L	4	Increment by One or Two	F=S+1+C_n	F→Y	Input	Input	Parity	Hold	Hi-Z	Hi-Z	L
L H L H	5	Sign/Magnitude-Two's Complement	F=$\overline{S}$+C_n if Z=L F=$\overline{S}$+C_n if Z=H	F→Y (Note 3)	Input	Input	Parity	Hold	Hi-Z	Hi-Z	L
L H H L	6	Two's Complement Multiply, Last Cycle	F=S+C_n if Z=L F=S−R−1+C_n if Z=H	Log. F/2→Y (Note 2)	Hi-Z	Input	F_0	Log. Q/2→Q	Input	Q_0	L
H L L L	8	Single Length Normalize	F=S+C_n	F→Y	F_3	F_3	Hi-Z	Log. 2Q→Q	Q_3	Input	L
H L H L	A	Double Length Normalize and First Divide Op.	F=S+C_n	Log 2F→Y	$R_3 \forall F_3$	F_3	Input	Log. 2Q→Q	Q_3	Input	L
H H L L	C	Two's Complement Divide	F=S+R+C_n if Z=L F=S−R−1+C_n if Z=H	Log. 2F→Y	$\overline{R_3 \forall F_3}$	F_3	Input	Log. 2Q→Q	Q_3	Input	L
H H H L	E	Two's Complement Divide, Correction and Remainder	F=S+R+C_n if Z=L F=S−R−1+C_n if Z=H	F→Y	F_3	F_3	Hi-Z	Log. 2Q→Q	Q_3	Input	L

NOTES: 1. At the most significant slice only, the C_{n+4} signal is internally gated to the Y_3 output.
2. At the most significant slice only, $F_3 \forall$ OVR is internally gated to the Y_3 output.
3. At the most significant slice only, $S_3 \forall F_3$ is generated at the Y_3 output.
4. Op codes 1, 3, 7, 9, B, D, and F are reserved for future use.

L = LOW
H = HIGH
X = Don't Care

Hi-Z = High Impedance
$\forall$ = Exclusive OR
Parity = $SIO_3 \forall F_3 \forall F_2 \forall F_1 \forall F_0$

[a]*Copyright © 1979 Advanced Micro Devices, Inc. Reproduced with permission of copyright owner.*

arithmetic shift avoids the sign bit position of the arithmetic expression, that is, the most significant bit of the most significant slice. On the other hand, in a logical shift operation, there is no "sign" bit, and the shift takes place through such a bit as well.

The SIO_0 and SIO_3 pins can be used to extend the sign of an arithmetic expression in the computer across the boundaries of individual 2903 chips. The SIO_0 can be used as a sign input, and SIO_3 as a sign output, under instruction control.

There are more instruction possibilities when I_0, I_1, I_2, I_3, or I_4 is not equal to zero. The definition of such instructions again depends on the definition of the $I_8, \ldots I_5$ bits and is presented in Table 6.7.

Note that the Y output of the 2901 has been extended in the 2903 to include more complex output combinations. Table 6.7 does not merely define the 4-bit Y output, but considers each of the four bits: Y_3, Y_2, Y_1, and Y_0. The output is different depending on whether one is referring to the most significant slice (which contains the sign bit in an arithmetic expression) or to other slices (which do not contain the sign bit).

The 2903 also provides three-state outputs. The high impedence ("Hi-Z") third state is shown for a number of signals in Tables 6.6 and 6.7.

The ALU destination control function for the 2903 is fairly complex, and it would be worthwhile to simplify the explanation for better understanding. There are five possible operations:

- Single length shift.
- Double length shift.
- Q-shift.
- Load.
- Sign extend.

The single length and double length shift may be done either arithmetically ("arith." in Table 6.7), or logically ("log." in Table 6.7.). The shift may also be either up or down (i.e., 2F → Y, or F/2 → Y). The single length shift does not utilize the Q register, so the Q register is unchanged (i.e., "hold"). The double length shift does use the Q register, and the Q register performs a logical shift, either up or down, corresponding to the instruction.

The Q shift operation, without writing the RAM, is another possibility. Only logical shifts, however, can be performed with the Q register.

Loading is yet another operation. The following possibilities exist:

LOAD
RAM
RAM and Q
Q
None

TABLE 6.7. AMD 2903 Destination Control[a]

I_8	I_7	I_6	I_5	Hex Code	ALU Shifter Function	SIO_3		Y_3		Y_2		Y_1	Y_0	SIO_0	$\overline{\text{Write}}$	Q Reg & Shifter Function	QIO_3	QIO_0
						Most Sig. Slice	Other Slices	Most Sig. Slice	Other Slices	Most Sig. Slice	Other Slices							
L	L	L	L	0	Arith. F/2→Y	Input	Input	F_3	SIO_3	SIO_3	F_3	F_2	F_1	F_0	L	Hold	Hi-Z	Hi-Z
L	L	L	H	1	Log. F/2→Y	Input	Input	SIO_3	SIO_3	F_3	F_3	F_2	F_1	F_0	L	Hold	Hi-Z	Hi-Z
L	L	H	L	2	Arith. F/2→Y	Input	Input	F_3	SIO_3	SIO_3	F_3	F_2	F_1	F_0	L	Log. Q/2→Q	Input	Q_0
L	L	H	H	3	Log. F/2→Y	Input	Input	SIO_3	SIO_3	F_3	F_3	F_2	F_1	F_0	L	Log. Q/2→Q	Input	Q_0
L	H	L	L	4	F→Y	Input	Input	F_3	F_3	F_2	F_2	F_1	F_0	Parity	L	Hold	Hi-Z	Hi-Z
L	H	L	H	5	F→Y	Input	Input	F_3	F_3	F_2	F_2	F_1	F_0	Parity	H	Log. Q/2→Q	Input	Q_0
L	H	H	L	6	F→Y	Input	Input	F_3	F_3	F_2	F_2	F_1	F_0	Parity	H	F→Q	Hi-Z	Hi-Z
L	H	H	H	7	F→Y	Input	Input	F_3	F_3	F_2	F_2	F_1	F_0	Parity	L	F→Q	Hi-Z	Hi-Z
H	L	L	L	8	Arith. 2F→Y	F_2	F_3	F_3	F_2	F_1	F_1	F_0	SIO_0	Input	L	Hold	Hi-Z	Hi-Z
H	L	L	H	9	Log. 2F→Y	F_3	F_3	F_2	F_2	F_1	F_1	F_0	SIO_0	Input	L	Hold	Hi-Z	Hi-Z
H	L	H	L	A	Arith. 2F→Y	F_2	F_3	F_3	F_2	F_1	F_1	F_0	SIO_0	Input	L	Log. 2Q→Q	Q_3	Input
H	L	H	H	B	Log. 2F→Y	F_3	F_3	F_2	F_2	F_1	F_1	F_0	SIO_0	Input	L	Log. 2Q→Q	Q_3	Input
H	H	L	L	C	F→Y	F_3	F_3	F_3	F_3	F_2	F_2	F_1	F_0	Hi-Z	H	Hold	Hi-Z	Hi-Z
H	H	L	H	D	F→Y	F_3	F_3	F_3	F_3	F_2	F_2	F_1	F_0	Hi-Z	H	Log. 2Q→Q	Q_3	Input
H	H	H	L	E	SIO_0→Y_0, Y_1, Y_2, Y_3	SIO_0	SIO_0	SIO_0	SIO_0	SIO_0	SIO_0	SIO_0	SIO_0	Input	L	Hold	Hi-Z	Hi-Z
H	H	H	H	F	F→Y	F_3	F_3	F_3	F_3	F_2	F_2	F_1	F_0	Hi-Z	L	Hold	Hi-Z	Hi-Z

Parity = $F_3 \forall F_2 \forall F_1 \forall F_0 \forall SIO_3$ L = LOW Hi-Z = High Impedance

$\forall$ = Exclusive OR H = HIGH

[a]*Copyright © 1979 Advanced Micro Devices, Inc. Reproduced with permission of copyright owner.*

TABLE 6.8. AMD 2903 ALU Destination Control Summary[a]

OPERATION		ALU SHIFTER	RAM WRITE	Q
SINGLE LENGTH SHIFT		UP DOWN ARITH UP ARITH DOWN	YES	NC
DOUBLE LENGTH SHIFT		UP DOWN ARITH UP ARITH DOWN	YES	UP DOWN UP DOWN
Q-SHIFT		PASS	NO	UP DOWN
LOAD	RAM RAM & Q Q NONE	PASS	YES YES NO NO	NC LOAD LOAD NC
SIGN EXTEND		SIO_0	YES	NC

NC = No Change

[a]*Copyright © 1979 Advanced Micro Devices, Inc. Reproduced with permission of copyright owner.*

Finally, the destination control function includes the sign extend operation. This instruction ($I_8, \ldots I_5$ = 1 1 1 0) extends the SIO_0 sign input to $Y_3, \ldots Y_0$.

The various 2903 ALU destination control functions can therefore be most simply summarized by Table 6.8.

We note above that the 2903 responds to certain instructions depending on whether it is the most significant slice or not in a multiple slice configuration. It is worthwhile therefore, to ask how the individual 2903 chip knows whether it is the most significant slice or not. The answer can be obtained by examining a typical 16-bit CPU configuration using the 2903, and using the 2902 for carry lookahead, as shown in Figure 6.6.

It is noted from Figure 6.6 that the least significant slice has the LSS pin ties to ground. When the LSS and the WRITE/MSS (most significant slice) pins are both tied HIGH (i.e., to +5), the slice operates as an intermediate slice. Finally, when WRITE/MSS (or W/MSS in Figure 6.6) is tied to the ground, the slice operates as the most significant slice.

Motorola 10800

The Motorola 10800 is a 4-bit bipolar microprocessor slice implemented in ECL technology and packaged in a 48-pin package. The functional block and pin diagram of the 10800 is shown in Figure 6.7.

As the block diagram indicates, the 10800 has three 4-bit independent data ports: Ø, I, and A. The Ø bus and the I bus are input/output ports, while the A bus is an input bus. Also on the chip are an ALU, a shift network, an accumulator, and I/O structures.

Control of the 10800 is achieved through the use of 16 select lines ASO–AS15. There are also five ALU status output lines, corresponding to indications of propagate, generate, carry out, overflow, and zero detect, and to parity output lines.

The ALU functions that can be executed by the 10800 are given in Table 6.9.

A more highly simplified version of the block diagram of the 10800 is shown in Figure 6.8 to highlight the important features.

The 10800 microprocessor slice is implemented in a system with the following supporting devices as shown in Figure 6.9:

- 10801, Microprogram Control Function Unit.
- 10802, Timing Function Unit.
- 10803, Memory Interface Function Unit.

These devices are then combined with ECL memories to produce the complete processor.

The operation of the 10800 system shown in Figure 6.9 is similar to that of other bit-slice microprogrammed systems. Instructions are executed by sequencing through the microprogram control memory. The 10801, shown in block diagram form in Figure 6.10, functions to perform the microprogram storage addressing.

The 10801 provides the required sequencing logic from its internal status register, instruction register, subroutine LIFO, address register, retry register, and incrementer.

The 10801 also provides means for generating the next control memory address. The next address logic block selects one of 16 instructions as determined by the I0 through I3 inputs. The possible instructions are given in Table 6.10.

The 10802 Timing Function Unit, shown in Figure 6.11, generates clock phases and provides start/stop capability and diagnostics. The start synchronizer circuit synchronizes an asynchronous start input signal to the master clock signal. The control section manages the start–stop mode and diagnostic operations. The signal cycle/single phase operation, for example, allows the system to advance by either one phase or one clock cycle for each start signal.

The 10803 Memory Interface Function Unit is shown in Figure 6.12. The Memory Interface Function Unit performs the memory addressing and data transfer operations, thereby freeing the 10800 for internal data processing operations. The figure shows the memory address register (MAR), the MAR multiplexer (MAR Mux), a four-word register file containing various memory addressing parameters (such as the program counter, index register, and so on), and a data matrix unit that defines the specific type of data transfer that

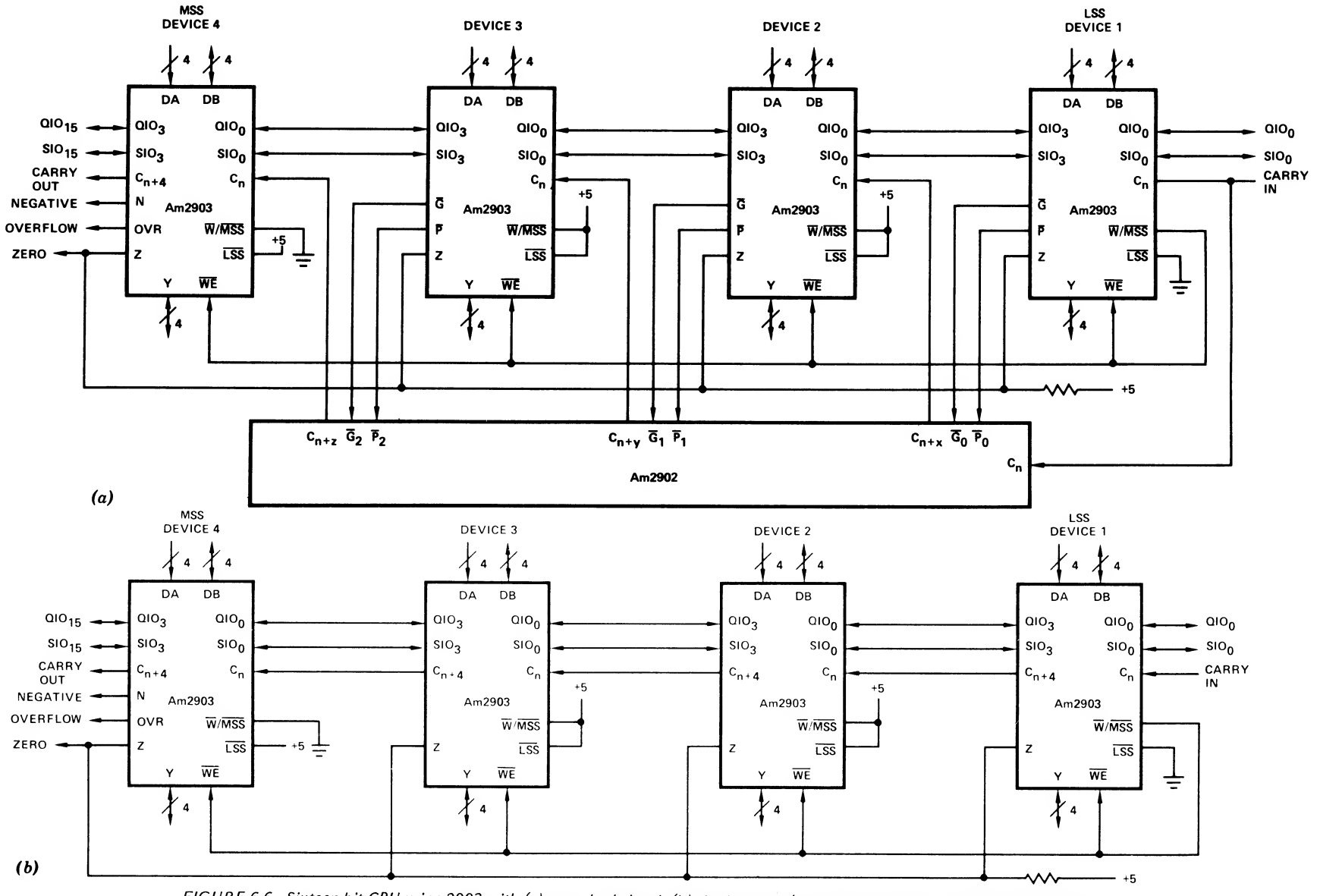


FIGURE 6.6 Sixteen-bit CPU using 2903 with (a) carry lookahead, (b) ripple carry. (Copyright © 1979 Advanced Micro Devices, Inc. Reproduced by permission of copyright owner.)

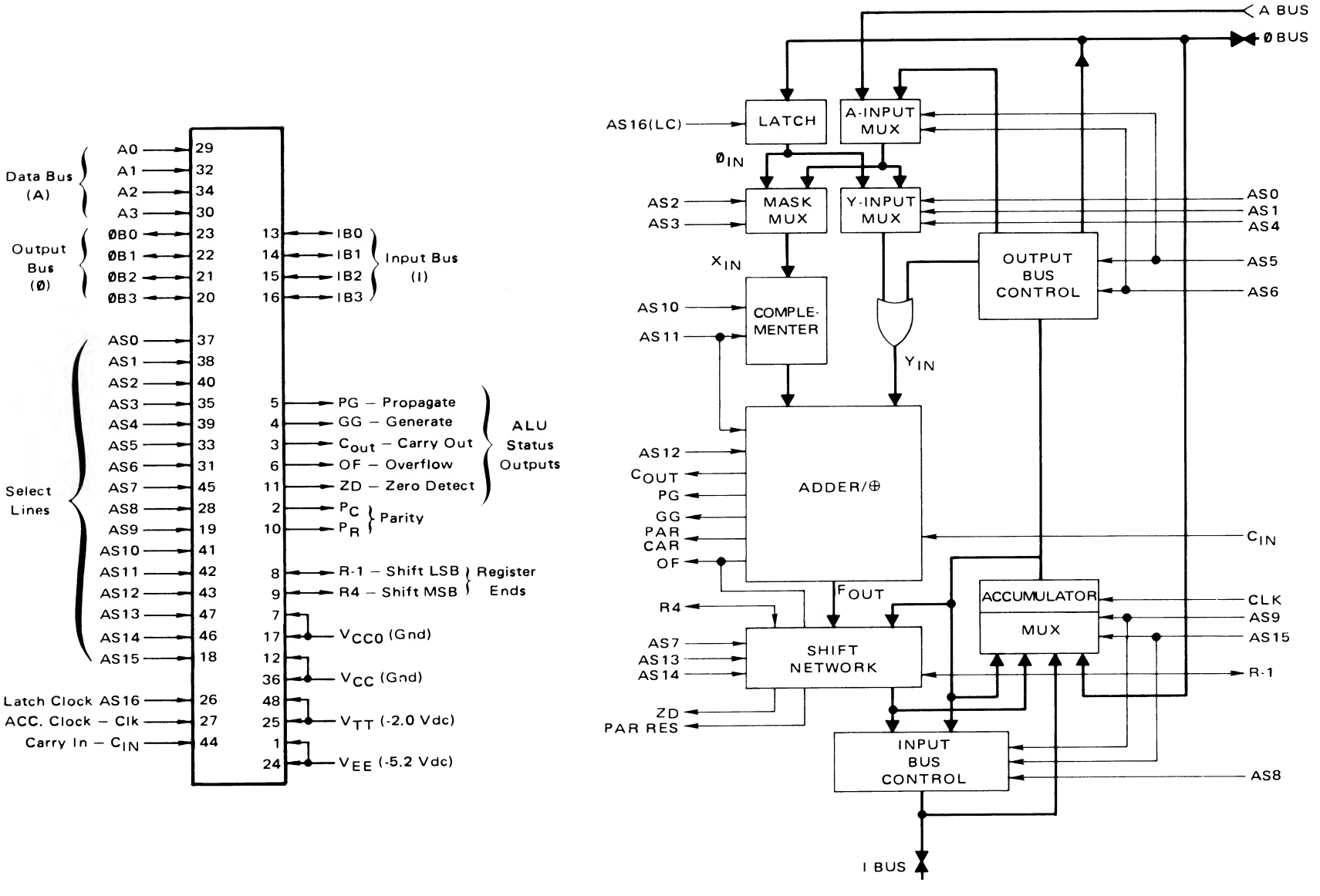


FIGURE 6.7 Motorola 10800 functional block diagram. (Courtesy Motorola Semiconductor Products, Incorporated.)

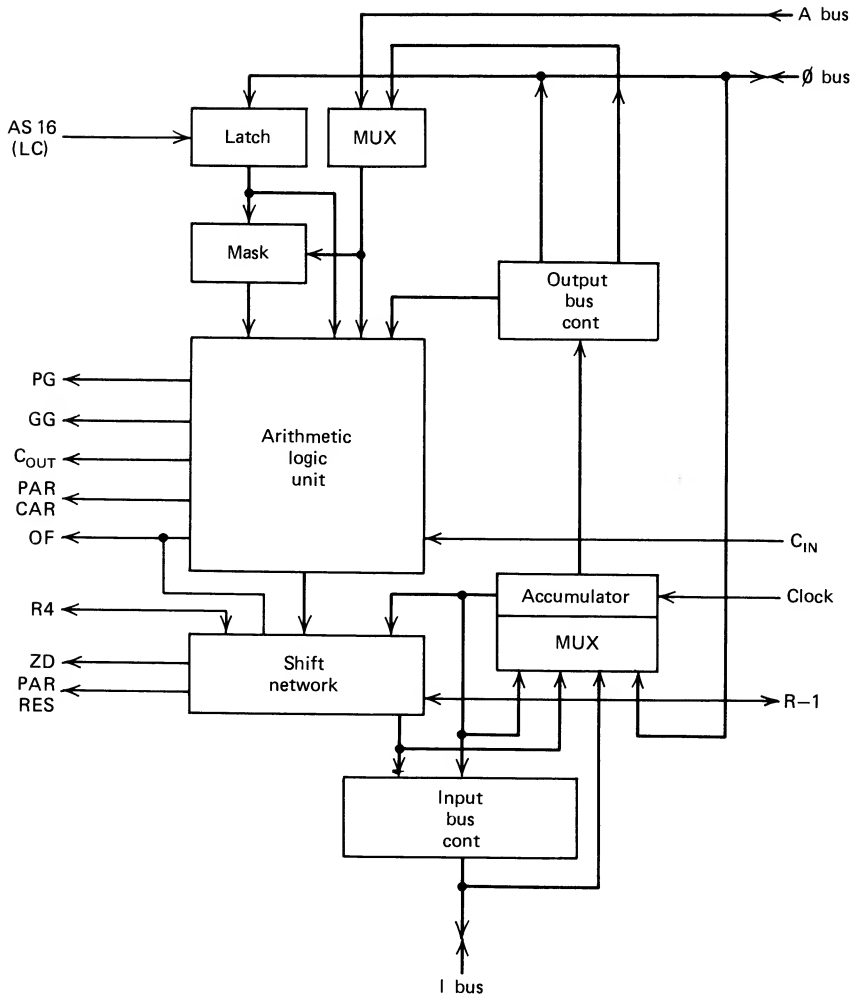


FIGURE 6.8 Motorola 10800 simplified block diagram. (Courtesy Motorola Semiconductor Products, Incorporated.)

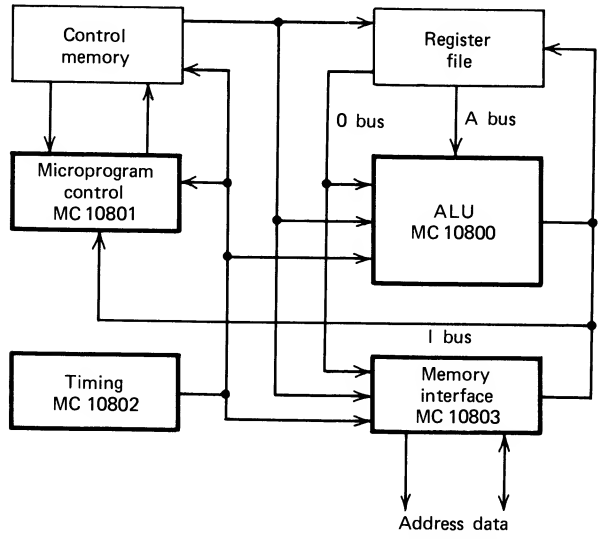


FIGURE 6.9 Motorola 10800 system. (Courtesy Motorola Semiconductor Products, Incorporated.)

TABLE 6.9. Motorola 10800 ALU Functions[a]

LOGIC	BINARY	BCD
F = Logic 0	F = A plus Ø	F = A plus Ø
F = A	F = A minus Ø	F = A minus Ø
F = Ø	F = Ø minus A	F = Ø minus A
F = $\overline{A}$	F = A	F = A
F = $\overline{Ø}$	F = Ø	F = Ø
F = A + Ø	F = $\overline{A}$	F = 9's Comp. A
F = A + $\overline{Ø}$	F = $\overline{Ø}$	F = 9's Comp. Ø
F = $\overline{A}$ + Ø	F = A minus 1	F = A plus 2
F = A • Ø	F = Ø minus 1	F = Ø plus 2
F = A • $\overline{Ø}$	F = A minus 2	F = A plus A
F = $\overline{A}$ • Ø	F = Ø minus 2	F = Ø plus Ø
F = A ⊕ Ø	F = A plus 2	F = ACC plus A
F = A ⊕ $\overline{Ø}$	F = Ø plus 2	F = ACC plus Ø
F = $\overline{A • Ø}$	F = A plus A	F = ACC minus A
F = $\overline{A}$ + $\overline{Ø}$	F = Ø plus Ø	F = ACC minus Ø
F = Logic 1	F = ACC plus A	F = ACC plus A • Ø
F = ACC • $\overline{A}$	F = ACC minus A	F = ACC minus A • Ø
F = ACC • $\overline{Ø}$	F = ACC plus Ø	F = ACC plus A + Ø
F = $\overline{ACC}$ + A	F = ACC minus Ø	F = ACC minus A + Ø
F = $\overline{ACC}$ + Ø	F = ACC plus A • Ø	
F = ACC ⊕ A	F = ACC minus A • Ø	
F = ACC ⊕ $\overline{A}$	F = ACC plus A + Ø	
F = ACC ⊕ Ø	F = ACC minus A + Ø	
F = ACC ⊕ $\overline{Ø}$		
F = ACC ⊕ $\overline{A • Ø}$		
F = ACC ⊕ $\overline{A}$ • Ø		
F = ACC ⊕ $\overline{A + Ø}$		
F = ACC ⊕ $\overline{A}$ + Ø		

[a] *Courtesy Motorola Semiconductor Products, Incorporated.*

TABLE 6.10. 10801 Instructions

INC	Increment
JMP	Jump to NA Inputs
JIB	Jump to I Bus
JIN	Jump to I Bus and Load R2
JPI	Jump to R2
JEP	Jump to External Bus (Ø Bus)
JL2	Jump to NA Inputs and Load R2
JLA	Jump to NA Inputs and Load R1
JSR	Jump to Subroutine
RTN	Return from Subroutine
RSR	Repeat Subroutine (Load R1 from NA Inputs)
RPI	Repeat Instruction (Jump to R1)
BRC	Branch to NA Inputs on Condition Otherwise Increment
BSR	Branch to Subroutine on Condition Otherwise Increment
ROC	Return from Subroutine on Condition or Jump to NA inputs
BRM	Branch and Modify Address with Branch Inputs (Multiway Branches)

is to occur. The data matrix is connected to the memory data register (MDR), which stores the incoming or outgoing data.

Finally, Figure 6.13 shows how the 10800 family of parts may be connected in a highly simplified block diagram to form a 16-bit processor system.

Texas Instruments SN 54S/74S-481

The SN 54S481 and SN 74S481 are 4-bit bipolar microprocessor slices using Schottky bipolar technology and packaged in a 48-pin package. The functional block diagram of the 54S/74S-481 is shown in Figure 6.14.

As the block diagram indicates, the 54S/74S-481 features an input port A, an input/output port B, a 4-bit data output port, and a 4-bit address output port. The chip further includes:

- ALU with carry lookahead, magnitude, and overflow decision capability.
- Double length accumulator with sign-bit handling.
- Dual memory address generators on chip.

The ALU functions that are executed by the 54S/74S-481 are given in Table 6.11.

Fairchild F100220

The Fairchild F100220 family is an 8-bit slice family of chips, implemented using ECL bipolar technology announced in early 1980. The basic parts of the family are shown in Table 6.12.

The ADIU functions both as an ALU and a memory interface. The inputs to the ADIU include four instruction lines and a function modifier/special status line

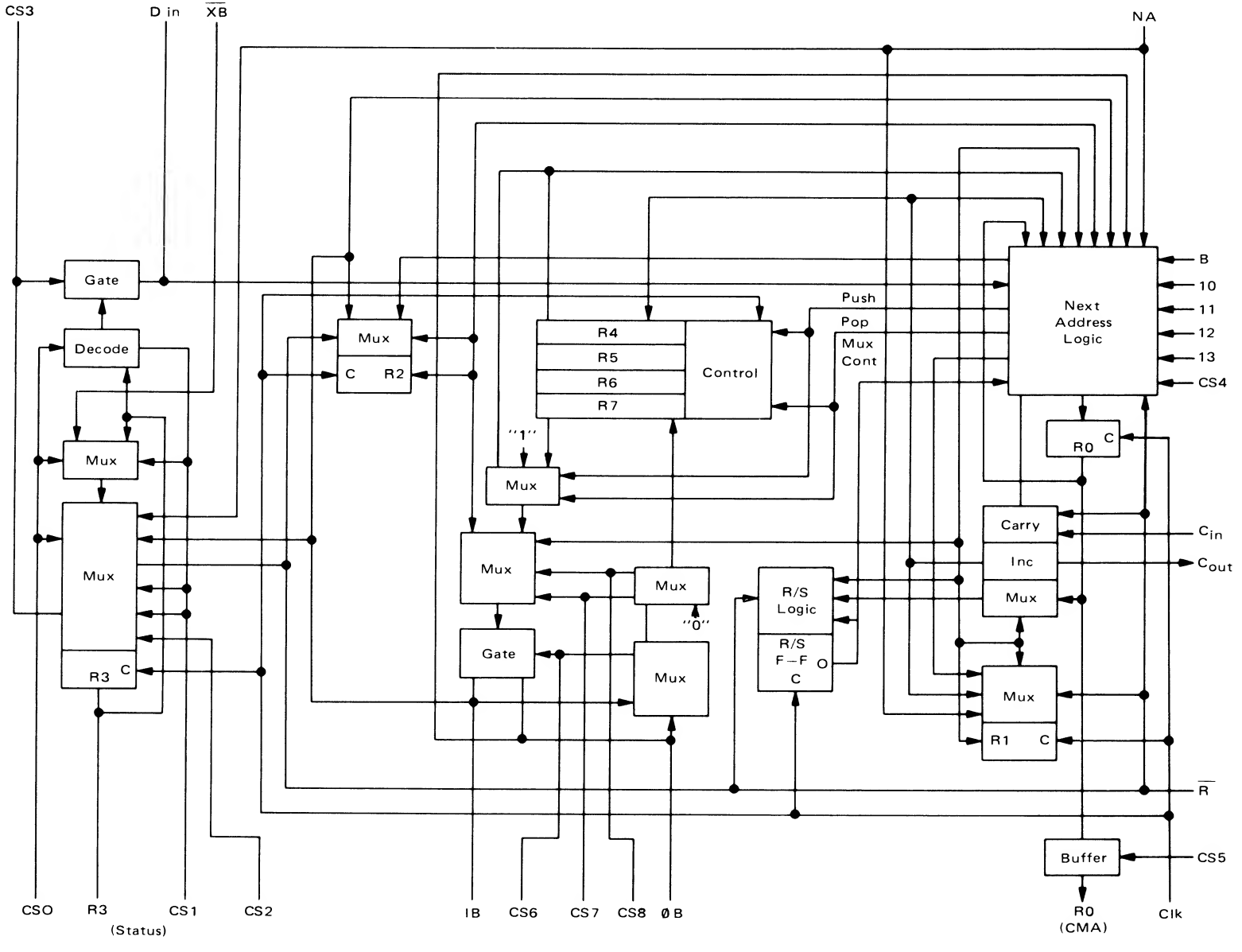


FIGURE 6.10 Motorola 10801 Microprogram Control Function unit block diagram. (Courtesy Motorola Semiconductor Products, Incorporated.)

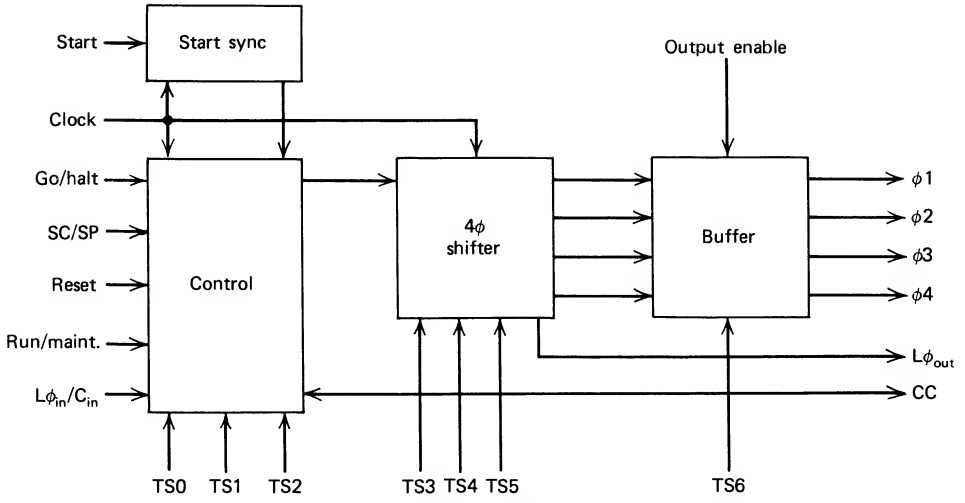


FIGURE 6.11 Motorola 10802 Timing Function unit block diagram. (Courtesy Motorola Semiconductor Products, Incorporated.)

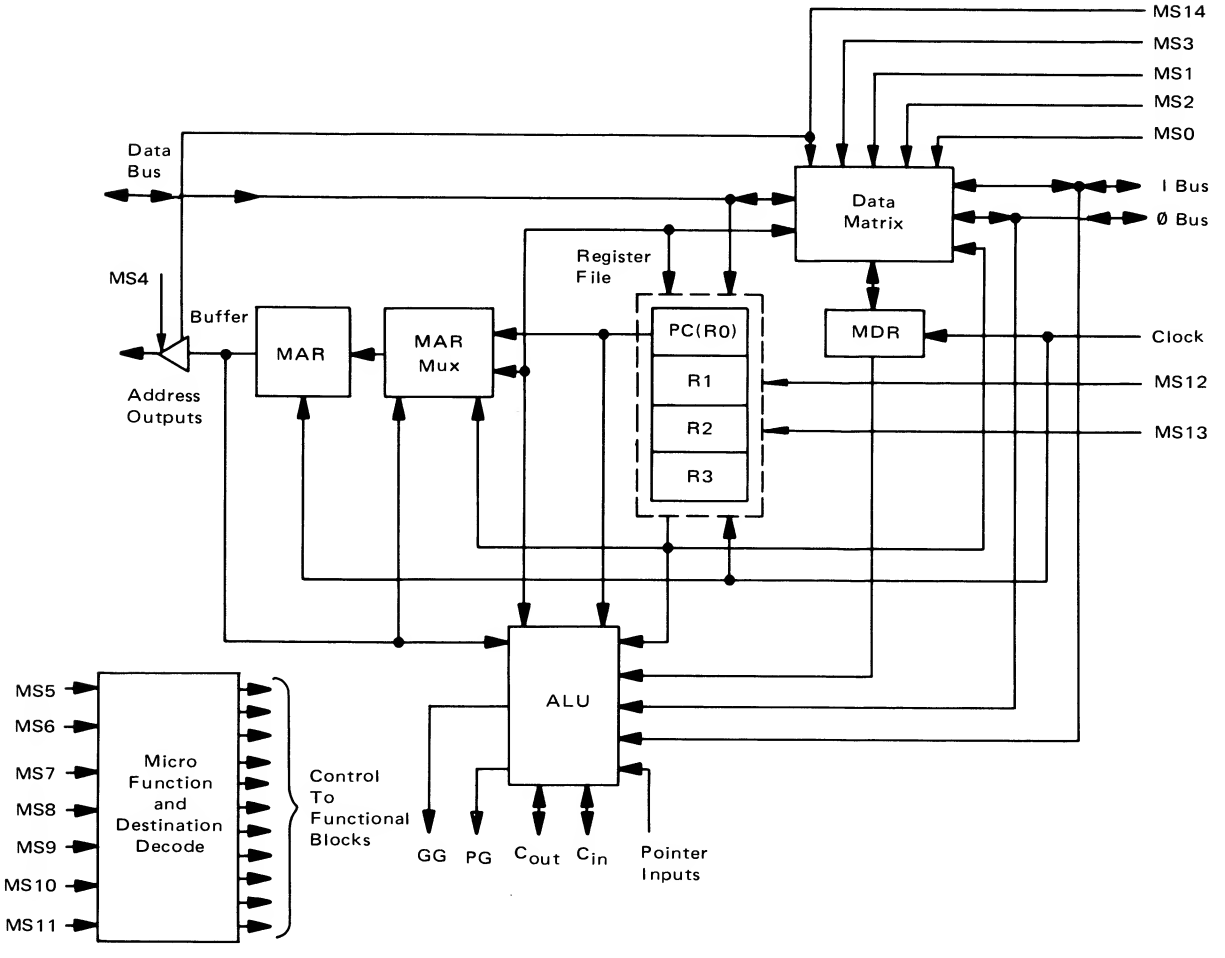


FIGURE 6.12 Motorola 10803 Memory Interface Function unit block diagram. (Courtesy Motorola Semiconductor Products, Incorporated.)

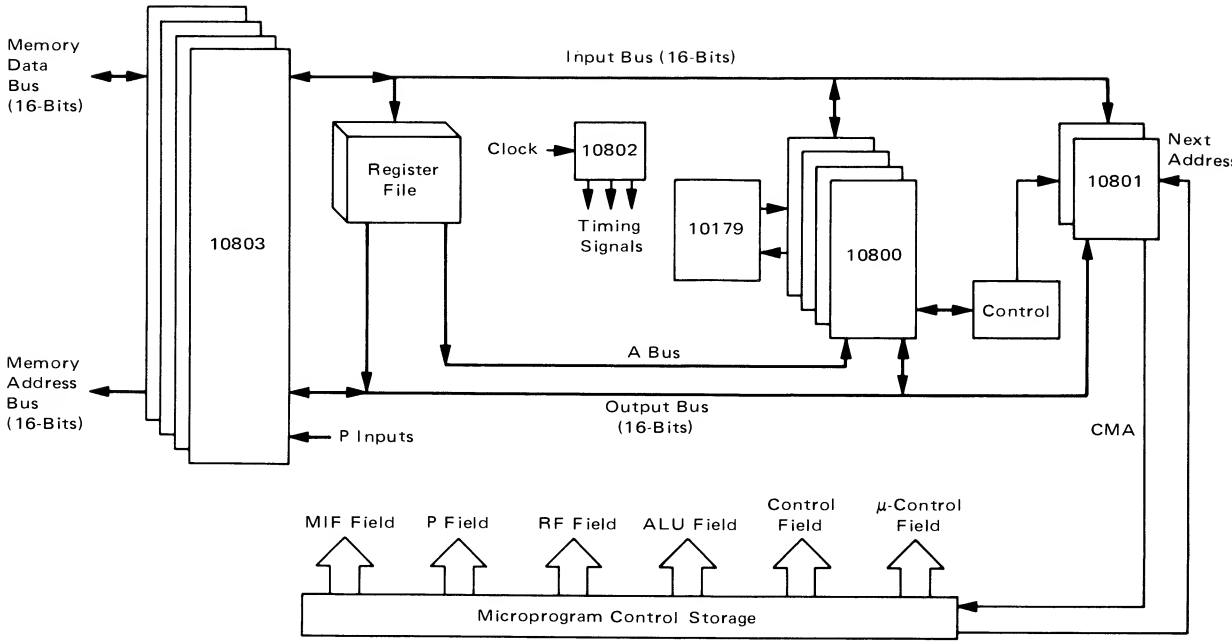


FIGURE 6.13 Motorola 10800 16-bit processor system. (Courtesy Motorola Semiconductor Products, Incorporated.)

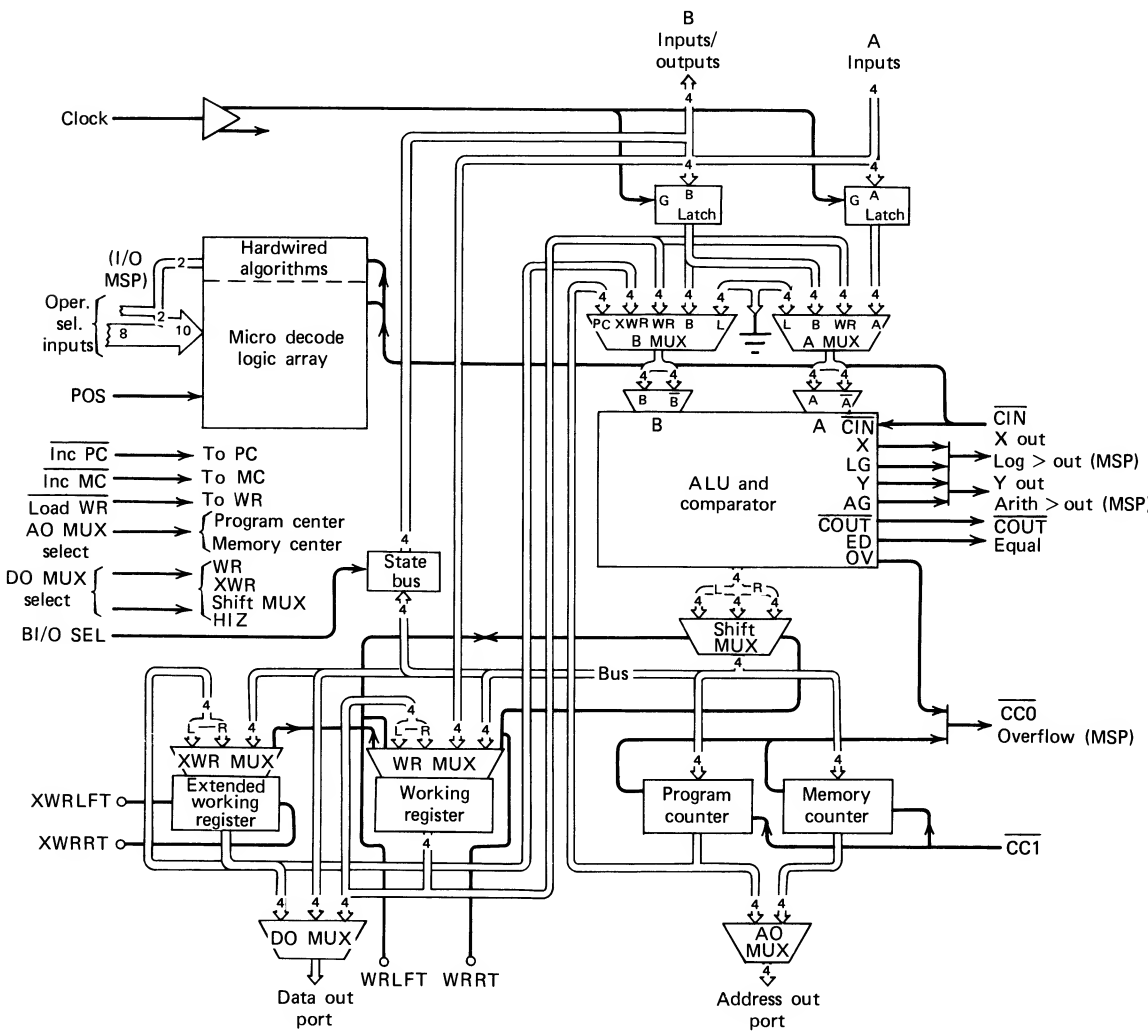


FIGURE 6.14 Texas Instruments 54S/74S-481. Functional block diagram. (Courtesy: Texas Instruments, Incorporated.)

TABLE 6.11. Texas Instruments 54S/74S-481 ALU Functions

Data Input		*Two's Complement Integer Arithmetic Op's*		*Logical Op's (Form VIII) $\overline{CIN}$ = H or L*		
A Port	*B Port*	$\overline{CIN} = L$	$\overline{CIN} = H$	*OR*	*NOR*	*EXNOR*
Zeros	Zeroes	1	0	Zeros	Ones	Ones
Zeros	Ones	0	Minus 1	Ones	Zeros	Zeros
Ones	Zeros	0	Minus 1	Ones	Zeros	Zeros
Ones	Ones	Minus 1	Minus 2	Ones	Zeros	Ones
A	Zeroes	A plus 1	A	A	$\overline{A}$	$\overline{A}$
$\underline{A}$	Ones	A	A minus 1	Ones	Zeros	A
$\overline{A}$	Zeroes	Minus A	Minus A minus 1	$\overline{A}$	A	$\underline{A}$
$\overline{A}$	Ones	Minus A minus 1	Minus A minus 2	Ones	Zeros	$\overline{A}$
Zeros	B	B plus 1	B	B	$\overline{B}$	$\overline{B}$
Ones	$\underline{B}$	B	B minus 1	Ones	Zeros	B
Zeros	$\overline{B}$	Minus B	Minus B minus 1	$\overline{B}$	B	$\underline{B}$
Ones	$\overline{B}$	Minus B minus 1	Minus B minus 2	Ones	Zeros	$\overline{B}$
A	$\underline{B}$	A plus B plus 1	A plus B	A +	$\overline{A} \cdot \overline{B}$	$\overline{A \oplus B}$
$\underline{A}$	$\overline{B}$	A minus B	A minus B minus 1	$\underline{A} + \overline{B}$	$\overline{A} \cdot \underline{B}$	$A \oplus B$
$\overline{A}$	$\underline{B}$	B minus A	B minus A minus 1	$\overline{A} + B$	$A \cdot \overline{B}$	$A \oplus B$
$\overline{A}$	$\overline{B}$	Minus A minus B minus 1	Minus A minus B minus 2	$\overline{A} + B$	$A \cdot B$	$\overline{A \oplus B}$

TABLE 6.12. Fairchild F100220 Family

Part Number	*Acryonym*	*Name*
F100220	ADIU	Address and data interface unit
F100221	DAS	Dual access stack
F100222	MFN	Multiple function network
F100223	PIU	Programmable interface unit
F100224	MSU	Microprogram sequencer unit

TABLE 6.13. F100220 ADIU Functions

I_3	I_2	I_1	I_0	*FMS*	*Hex Code*	*ADIU Functions*
L	L	L	L	$\frac{L}{H}$	0	No operation
L	L	L	H	$\frac{L}{H}$	1	C→R, ECC encode of R→A
L	L	H	L	$\frac{L}{H}$	2	A minus B→R (decimal unpacked) A minus B→R (decimal packed)
L	L	H	H	$\frac{L}{H}$	3	A plus B→R (decimal unpacked) A plus B→R (decimal packed)
L	H	L	L	$\frac{L}{H}$	4	B minus A→R (binary)
L	H	L	H	$\frac{L}{H}$	5	R zero→R R ECC decode of A→R
L	H	H	L	$\frac{L}{H}$	6	A minus B→R (binary)
L	H	H	H	$\frac{L}{H}$	7	A plus B→R (binary) C plus B→R (binary)
H	L	L	L	$\frac{L}{H}$	8	A shift left→R A shift right→R
H	L	L	H	$\frac{L}{H}$	9	B shift left→R B shift right→R
H	L	H	L	$\frac{L}{H}$	A	$\underline{A}$→R $\overline{A}$→R
H	L	H	H	$\frac{L}{H}$	B	$A \wedge B$→R $\overline{A \vee B}$→R
H	H	L	L	$\frac{L}{H}$	C	$\underline{B}$→R $\overline{B}$→R
H	H	L	H	$\frac{L}{H}$	D	$A \vee B$→R $\overline{A \wedge B}$→R
H	H	H	L	$\frac{L}{H}$	E	$\underline{C}$→R $\overline{C}$→R
H	H	H	H	$\frac{L}{H}$	F	$A \vee B$→R

(FMS), which together specify one of the 27 possible instructions shown in Table 6.13.

When the ADIU includes three bidirectional 9-bit buses (i.e., 8 data bits plus parity) that serve as sources for the ALU operand, as well as destinations for the ALU result.

When the ADIU functions as a memory interface it functions together with the MFN unit to support single bit error correction and double bit error detection.

7

AMD 2901 system design

As an example of system design using bit slice microprocessors, we consider a design using the AMD 2901. In particular we discuss the following issues:

- Microprogrammed design.
- System configuration.
- A basic computer controller.
- The instruction set.
- Software.

Microprogrammed Design

Microprogrammed design is concerned with the construction of a computer architecture for executing microinstructions. The microinstructions consist of the most basic machine operations, so that the set of microinstructions is equivalent to a machine instruction, such as the assembly language instructions considered in the preceding chapters dealing with the 16-bit microprocessors. The set of all microinstructions is stored in a *microprogram memory*. A specific sequence of microinstructions is called a *microprogram*. The subject of *microprogramming* is concerned with writing sequences of microinstructions for performing a particular operation.

Now that the basic definitions have been established, we can turn to specific examples. An example of a microinstruction (or microprogram instruction) is shown in Figure 7.1. The microinstruction is typically from 32 to 128 bits in length and consists of discrete fields that define elemental signals associated with specific machine functions. Typical examples of these discrete fields are shown in Figure 7.1: a branch address and instruction fields associated with specific machine elements. In the case of a 2903 system, the specific machine elements are devices such as the 2910 and 2903.

As we point out in the preceding chapter, the advantage of microprogrammed design is that the microinstruction word can be tailored to the user's application, and thus the architecture of the computer can be customized.

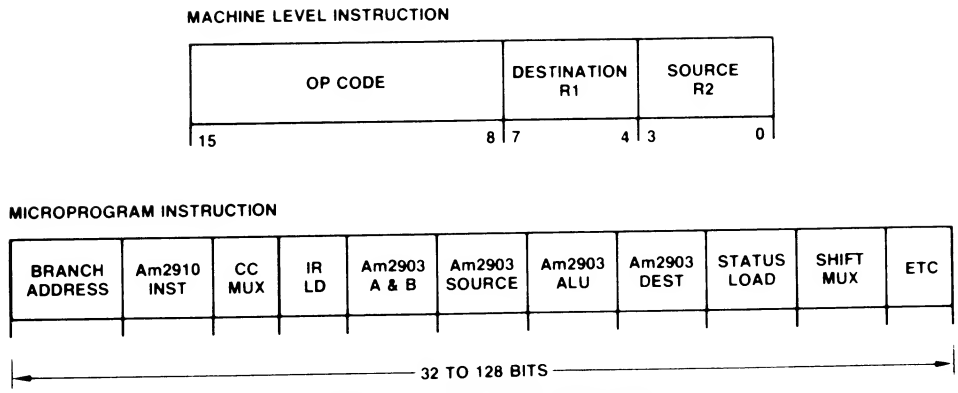


FIGURE 7.1 Microinstruction format.

Since the fields of the microinstruction are used to control the various machine devices, the architecture of the machine can be changed merely by changing microinstructions. As an example, the ALU source operand, the ALU function, the ALU destination, shift control, bus control, and other machine functions can be changed merely by changing the microcode.

There are basically two levels of system design: hardware and software. The hardware design is fixed by the components and their interconnections. Software design is concerned with the microprogram that operates with the hardware. Although one is constrained by the hardware design in a microprogrammed system much like in a regular computer system, there is more flexibility in the software design. On the other hand, the software design for a microprogrammed system is far more difficult, and requires a greater knowledge and understanding of the hardware components of the system than a normal computer. We therefore turn our attention first to the hardware design and system configuration.

System Configuration

We begin by illustrating some of the hardware configurations associated with microprogrammed design. Figure 7.2 shows the heart of a microprogrammed system. A machine level instruction is supplied over the data bus to an instruction register. The instruction register is connected to an instruction decoder, which is in turn connected to a sequencer or controller, such as the AMD 2910 and 2911.

The difference between the AMD 2910 and the 2909 or 2911 is that the 2910 is 12-bits wide, whereas the 2909 and 2911 are 4-bit slices. The AMD 2910 is thus able to address up to 4096 words of microcode. For applications with small or larger microprogram size, the 2911 slice microprogram sequence controller is still preferred.

A functional block diagram of the AMD 2910 is presented in Figure 7.3. The function of the microprogram sequence controller, as the name implies, is to control the sequence in which the microinstructions stored in the microprogram memory are executed. Since the architectures of the 2909 and 2911 are similar to that of the 2910, we only describe the 2910 here.

In the 2910 there are four possible address sources:

- The register/counter (R).

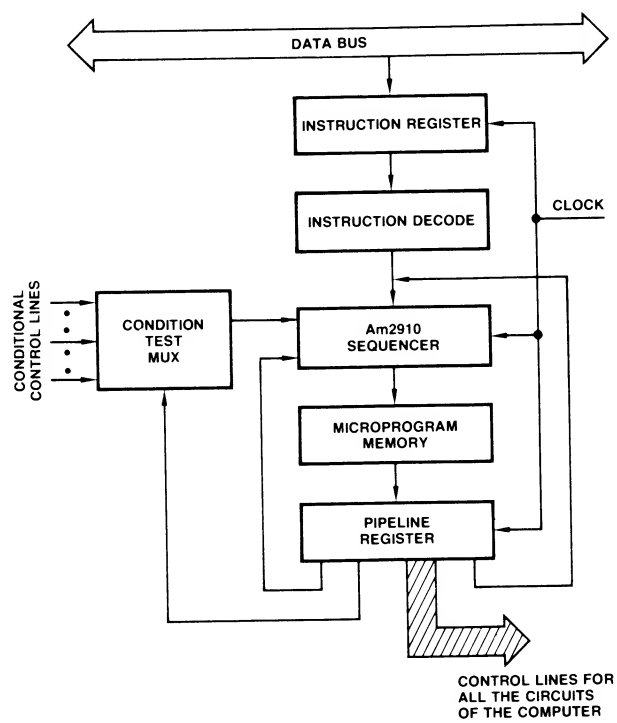


FIGURE 7.2 Microprogrammed system architecture.

Am2910 BLOCK DIAGRAM

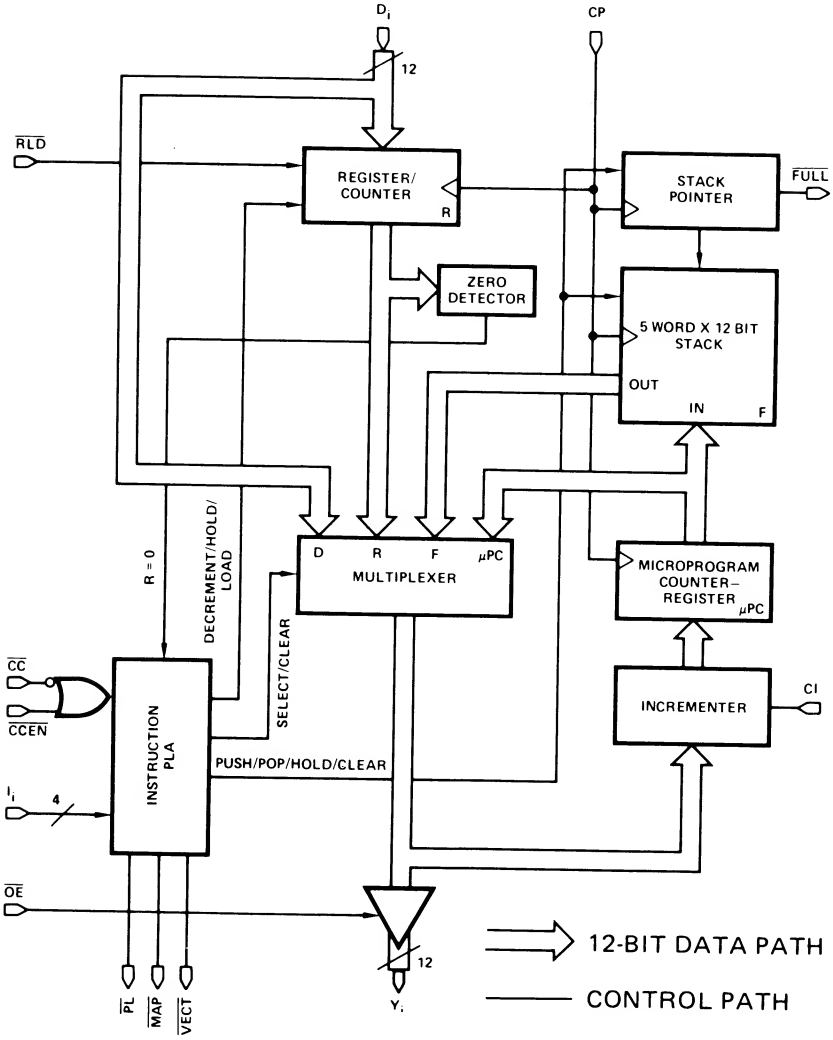


FIGURE 7.3 AMD 2910 functional block diagram. (Copyright © 1979 Advanced Micro Devices, Inc. Reproduced with permission of copyright owner.)

- Microprogram counter (PC).
- Stack (F).
- Direct input (D).

The multiplexer, shown in the center of the block diagram in Figure 7.3, selects the 12-bit address from one of these sources. The microprogram counter usually contains an address one greater than the present address and is used to execute sequential microinstructions. The stack input is used to provide a return address when executing loops or subroutines are being executed. The direct input is used as an address source when a branch is being performed.

The operation of the 2910 is controlled by the user by means of a 4-bit instruction provided on the instruction input I_i (i = 0, 1, 2, 3), which is connected to an instruction programmable logic array (PLA) shown in Figure 7.3. The 16 possible user instructions for the 2910 are given in Table 7.1.

A detailed explanation of the instruction set of the 2910, together with a comparison with the instruction set of similar microprogram sequence controllers, is presented later in this chapter.

"The output of the sequencer is connected to a microprogram memory, which stores the microinstructions. The output microinstruction is stored in a pipeline register, and various fields of the microinstruction are

TABLE 7.1 Am 2910 Instruction Set

HEX I_3-I_0	MNEMONIC	NAME	REG/ CNTR CON-TENTS	FAIL $\overline{CCEN}$ = LOW and $\overline{CC}$ = HIGH		PASS $\overline{CCEN}$ = HIGH or $\overline{CC}$ = LOW		REG/ CNTR	ENABLE
				Y	STACK	Y	STACK		
0	JZ	JUMP ZERO	X	0	CLEAR	0	CLEAR	HOLD	PL
1	CJS	COND JSB PL	X	PC	HOLD	D	PUSH	HOLD	PL
2	JMAP	JUMP MAP	X	D	HOLD	D	HOLD	HOLD	MAP
3	CJP	COND JUMP PL	X	PC	HOLD	D	HOLD	HOLD	PL
4	PUSH	PUSH/COND LD CNTR	X	PC	PUSH	PC	PUSH	Note 1	PL
5	JSRP	COND JSB R/PL	X	R	PUSH	D	PUSH	HOLD	PL
6	CJV	COND JUMP VECTOR	X	PC	HOLD	D	HOLD	HOLD	VECT
7	JRP	COND JUMP R/PL	X	R	HOLD	D	HOLD	HOLD	PL
8	RFCT	REPEAT LOOP, CNTR ≠ 0	≠ 0	F	HOLD	F	HOLD	DEC	PL
			= 0	PC	POP	PC	POP	HOLD	PL
9	RPCT	REPEAT PL, CNTR ≠ 0	≠ 0	D	HOLD	D	HOLD	DEC	PL
			= 0	PC	HOLD	PC	HOLD	HOLD	PL
A	CRTN	COND RTN	X	PC	HOLD	F	POP	HOLD	PL
B	CJPP	COND JUMP PL & POP	X	PC	HOLD	D	POP	HOLD	PL
C	LDCT	LD CNTR & CONTINUE	X	PC	HOLD	PC	HOLD	LOAD	PL
D	LOOP	TEST END LOOP	X	F	HOLD	PC	POP	HOLD	PL
E	CONT	CONTINUE	X	PC	HOLD	PC	HOLD	HOLD	PL
F	TWB	THREE-WAY BRANCH	≠ 0	F	HOLD	PC	POP	DEC	PL
			= 0	D	POP	PC	POP	HOLD	PL

Note 1: If $\overline{CCEN}$ = LOW and $\overline{CC}$ = HIGH, hold; else load. X = Don't Care

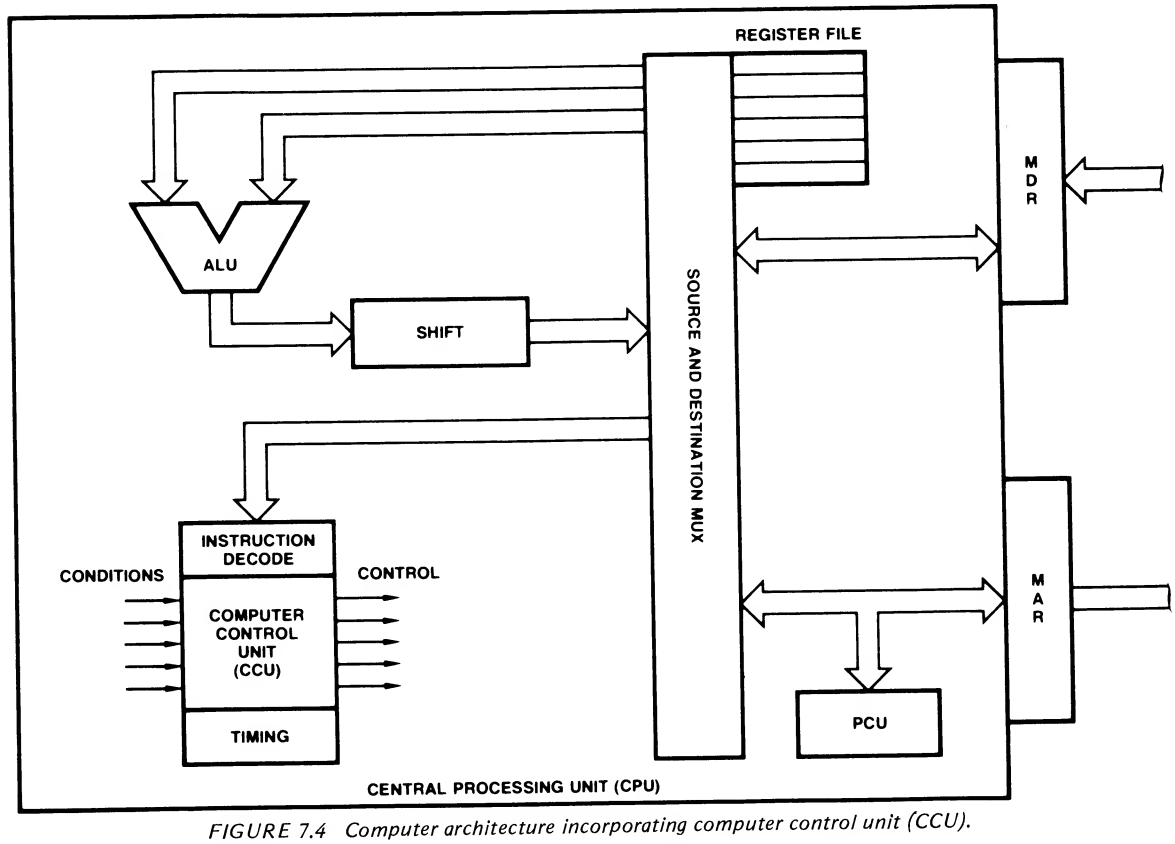


FIGURE 7.4 Computer architecture incorporating computer control unit (CCU).

routed to the appropriate machine devices for execution. Since such a configuration, as is shown in Figure 7.3 operates as a control unit, it is often referred to as the Computer Control Unit (CCU). The CCU forms a portion of a larger system consisting of an arithmetic logic unit and a register file such as shown in Figure 7.4.

"As the name implies, the function of the sequencer used is to supply a sequence of addresses to the microprogram memory. As we point out earlier, one of the features of the microinstruction is that it includes information on the succeeding microinstruction to be executed. More specifically, a "branch address" or branch instruction is provided within the microinstruction itself and is fed in parallel back to the microprogram address counter as "data." Such an operation is illustrated in a simplified block diagram of a portion of the microprogrammed system shown in Figure 7.5, the microprogram memory address controller.

"Another feature of the microinstruction suggested from Figure 7.5, is the "load select" field. This 2-bit field is used to control a multiplexer that has four single-bit inputs. Depending on the value of the 2-bit field in the microinstruction, one of the four inputs is selected. The selected input is then loaded into the microprogram address counter (and hence the name "load select"). These inputs to the multiplexer therefore provide a

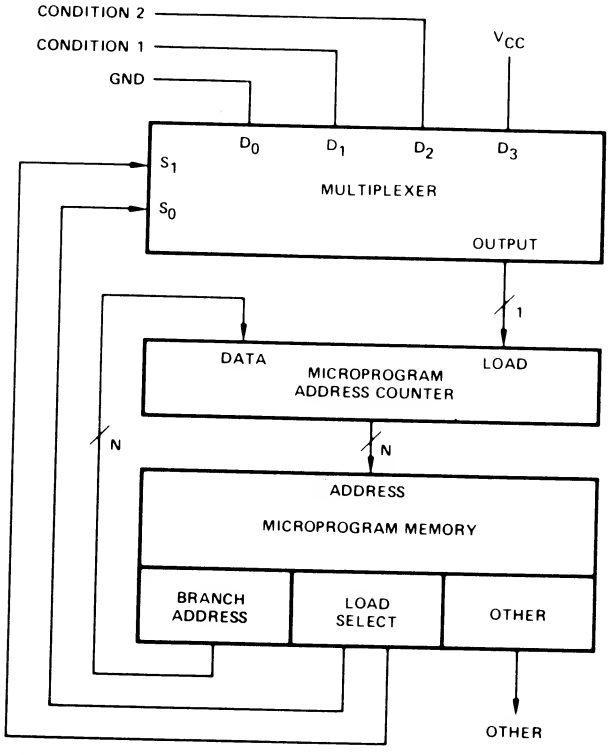


FIGURE 7.5 Microprogram memory address controller (simple version).

TABLE 7.2. Load Select Control Field

S_1	S_0	*Function*
0	0	Continue
0	1	Jump condition if D_1 input is true
1	0	Jump condition if D_2 input is true
1	1	Jump unconditionally

"conditional" capability to the controller in the sense that the previously supplied branch instruction is now a conditional branch instruction depending on the value loaded into the microprogram as counter."

The interpretation of the "load select" field is given in Table 7.2.

As Table 7.2 indicates, if the 2-bit load select field is 0 0, the multiplexer selects the D_0 or GROUND input, the microprogram address counter is inactive to branch, and the next sequential microinstruction is selected. The selection of the next sequential microinstruction is sometimes referred to as "continue."

If the 2-bit load select field is 0 1, the D_1 input is selected. If D_1 is false (i.e., logic 0), the microprogram address counter merely selects the next sequential microinstruction, that is, continues. If, however, the D_1 input is true (i.e., logic 1), a jump condition exists and the branch address is loaded in the next clock cycle.

Similarly, if the 2-bit load select field is 1 0, the D_2 input is selected. If D_2 is false, the microprogram address counter continues. If D_2 is true, the branch address is loaded into the microprogram address counter in the next clock cycle.

Finally, if the 2-bit load select field is 1 1, the D_3 input is selected. D_3 is tied to V_{cc}, which is logic 1. A logic 1 on the load input of the microprogram address counter automatically causes the branch address to be loaded into the counter. Thus the 1 1 load select code serves as an UNCONDITIONAL JUMP command.

The actual implementation of a microprogram memory address controller is somewhat more sophisticated than the simplified version presented in Figure 7.5. Figure 7.6 shows the more typical implementation of microprogram address control incorporating a pipeline register and address multiplexer. We see that the output of the microprogram memory is connected to a pipeline register. As we pointed out in the preceding chapter with reference to the 2901, the pipeline register permits the overlapping of the microinstruction being executed with the fetching of the next microinstruction. The address multiplexer at the input of the microprogram memory functions to select either the microprogram counter register or the pipeline register as the source of

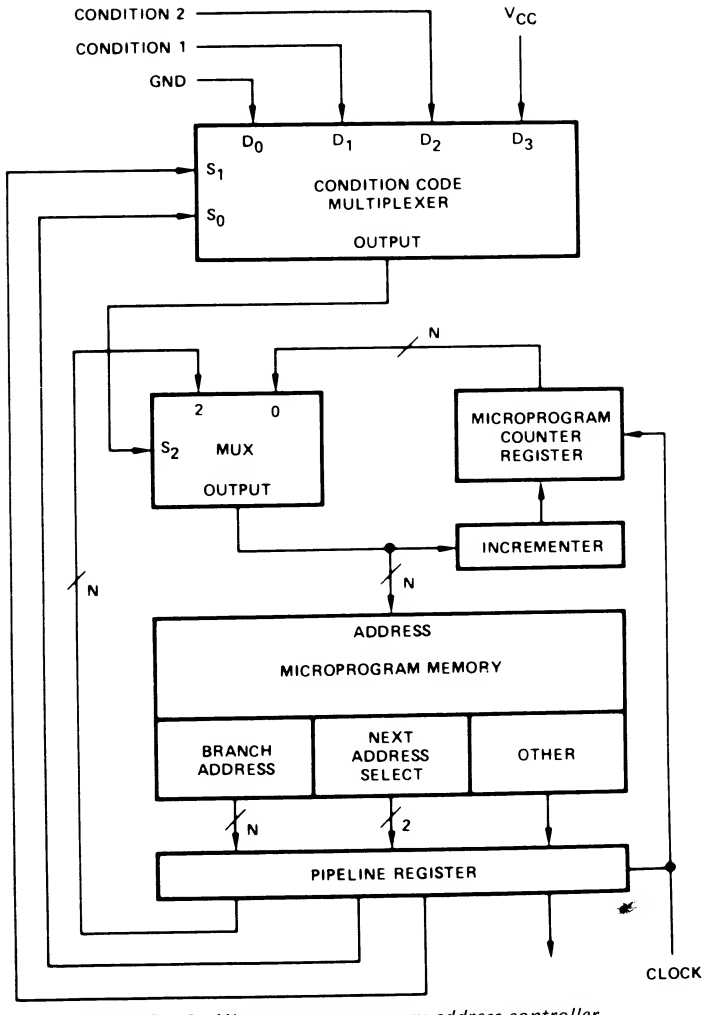


FIGURE 7.6 Microprogram memory address controller.

the next address for the microprogram memory. The condition code multiplexer, which replaces the multiplexer in Figure 7.5, has an output that controls the address multiplexer.

Another feature of the implementation of Figure 7.6 is the replacement of the microprogram address counter of Figure 7.5 with the microprogram counter register and incrementer. When the address multiplexer selects the microprogram counter register, the same effect is achieved as with the microprogram address counter of Figure 7.5.

A Basic Computer Controller

Having described the operation of the microprogram memory and the microprogram sequence controller, we next put the two concepts together to form a basic computer controller. Figure 7.7 shows a highly simplified block diagram of such a controller using the AMD 2910. In this figure the various internal elements of the 2910 have been simplified, emphasizing the origin of the address to be applied to the microprogram memory. The interface with the rest of the computer system is provided from the data bus and an output of the pipeline register. Connected to the data bus is an instruction register, which receives a machine level instruction (refer to Figure 7.1 for the fields in a machine level instruction). The op code portion of the machine level instruction is then decoded by means of a mapping PROM that functions to provide the first microinstruction corresponding to the machine level instruction. The executing microinstruction then provides the address of the next microinstruction, and the process continues.

The same type of computer controller can be implemented using the 2911 microprogram sequence con-

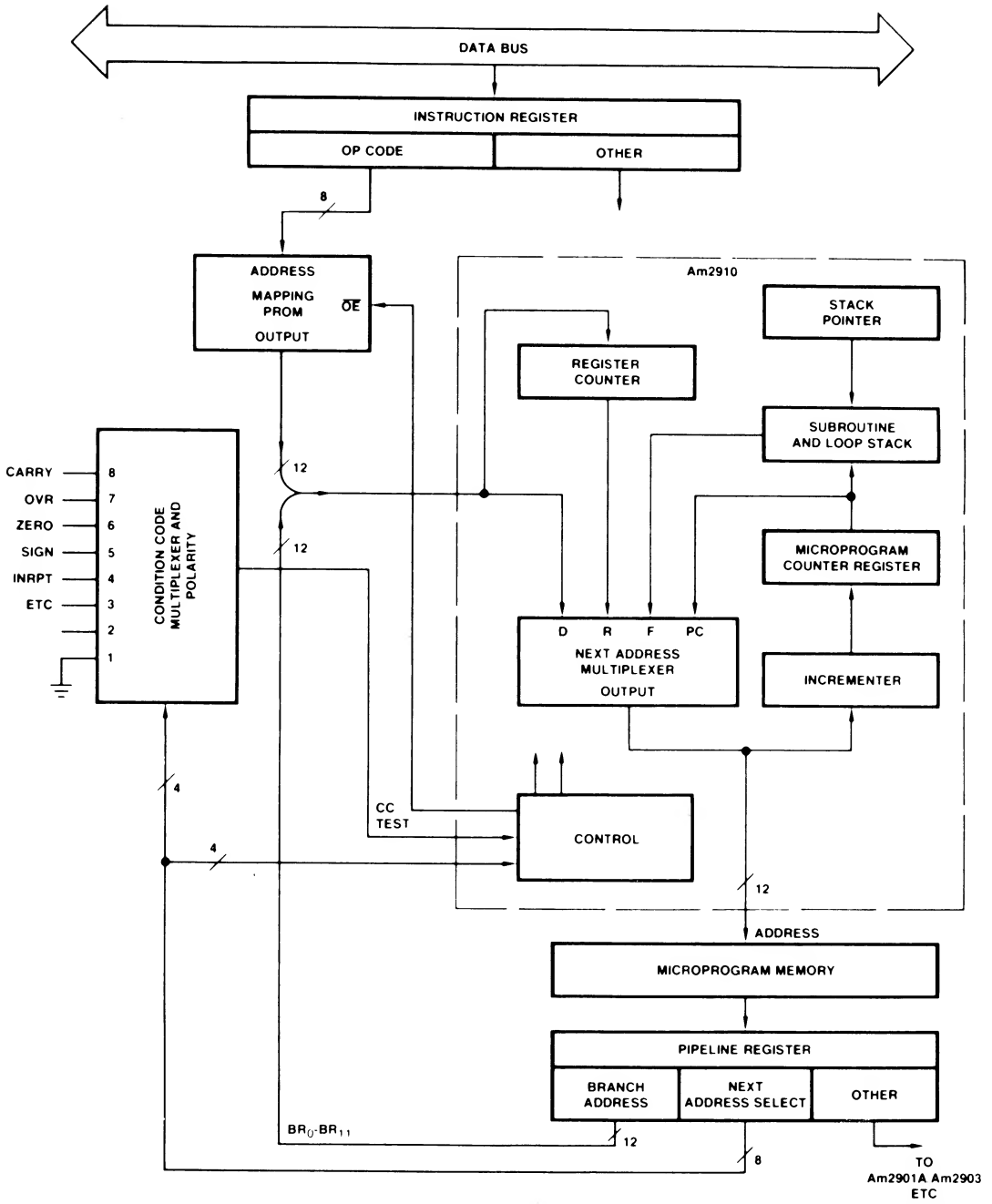


FIGURE 7.7 Computer control unit using the AMD 2910. (Copyright © 1979 Advanced Micro Devices, Inc. Reproduced with permission of copyright owner.)

troller and the 29811 next-Address Control Unit, as shown in Figure 7.8.

We now turn to the details of the implementation of the computer control unit using the AMD 2911 microprogram sequencer and the AMD 29811 next-Address Control, shown in Figure 7.8. Examining this implementation in greater detail, we note that the condition code multiplexer has been replaced by the 29811 next-address control unit. Like the 2910, the 2911 includes a stack pointer and a subroutine-and-loop stack. This feature of the 2911 permits subroutining to be implemented in microprogramming just as in regular programming.

Directing our attention to the 29811, we note that it has a 4-bit input from the pipeline register (i.e., from the branch address portion of the microinstruction executed) and a TEST input arising from the condition code multiplexer. Various conditions as inputs of the condition code multiplexer have now been explicitly named, such as "carry," "overflow," and "zero."

The 29811 has several outputs: 2 bits to the next-

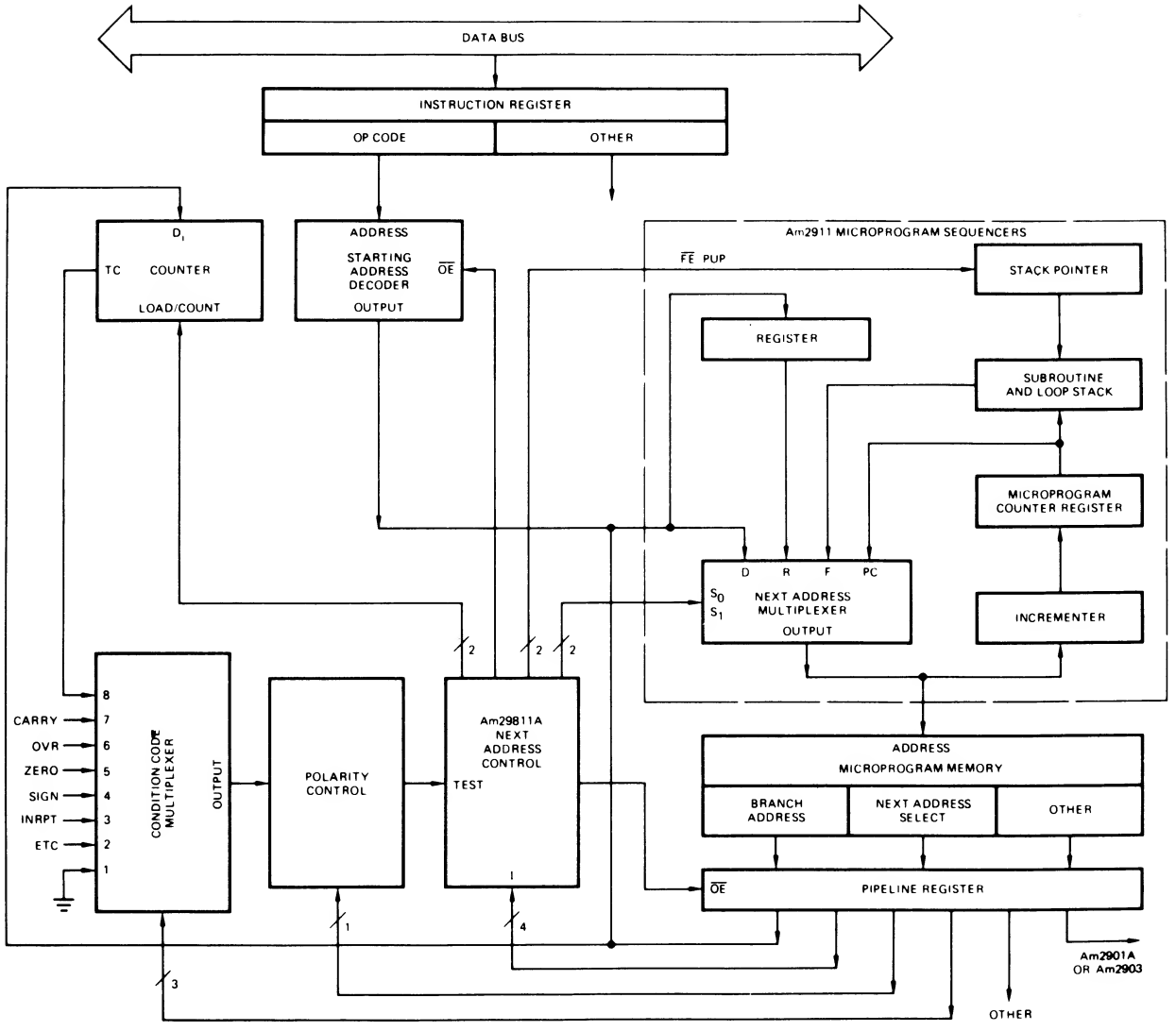


FIGURE 7.8 Computer control unit using AMD 2911 and 29811.

address multiplexer, the S_0 and S_1 signals we discuss earlier in connection with Figure 7.5 and Table 7.2; a 2-bit output to the stack pointer or "file" (labeled FE and PUP); a 1-bit output to the starting address decoder (OE); and a 2-bit output to the counter.

The 29811 is able to control the various units to which it is connected by means of its input code: the 4-bits from the branch address, plus the TEST input. The 29811 can therefore be said to have its own "instruction set" based on the value of the 4-bit input from the pipeline register, together with the TEST input. This instruction set is presented in Table 7.3, and a functional description including the various output alternatives is presented in Table 7.4.

A more detailed schematic diagram of the computer control unit using the 2910 is presented in Figure 7.9. The 25LS377 is used to form the 16-bit instruction register. The mapping PROM consists of three 29761 PROMs that are connected to the 2910. The condition code multiplexer is formed by two 2922s. Finally, the pipeline register is formed by U9, U10, U11, U12, and U13—a 74S175, a 74S174, and three 2918s. These devices are all edge-triggered D-type flip-flop registers that are specifically chosen for the properties required by their respective positions in the pipeline register. The tables in the left hand portion of Figure 7.9 indicate the various signal values on the labeled lines and their function.

A detailed schematic diagram of the computer control unit using the 2911 and 29811 devices is presented in Figure 7.10. To make the diagram more comparable with Figure 7.9, three 2911s were used so that a 12-bit

TABLE 7.3. 1—Am29811 Instruction Set

Mnemonic	I_3	I_2	I_1	I_0	*Instruction*
JZ	L	L	L	L	Jump to address zero
CJS	L	L	L	H	Conditional jump to subroutine with jump address in pipeline register
JMAP	L	L	H	L	Jump to address at mapping prom output
CJP	L	L	H	H	Conditional jump to address in pipeline register
PUSH	L	H	L	L	Push stack and conditionally load counter
JSRP	L	H	L	H	Jump to subroutine with starting address conditionally selected from Am2911 R register or pipeline register
CJV	L	H	H	L	Conditional jump to vector address
JRP	L	H	H	H	Jump to address conditionally selected from Am2911 or register or pipeline register
RFCT	H	L	L	L	Repeat at loop if counter is not equal to zero
RPCT	H	L	L	H	Repeat pipeline address if counter is not equal to zero
CRTN	H	L	H	L	Conditional return from subroutine
CJPP	H	L	H	H	Conditional jump to pipeline address and pop stack
LDCT	H	H	L	L	Load counter and continue
LOOP	H	H	L	H	Test end of loop
CONT	H	H	H	L	Continue to next address
JP	H	H	H	H	Jump to pipeline register address

TABLE 7.4. Functional Description of Am 29811 Instruction Set

MNEMONIC	INPUTS			OUTPUTS				
	INSTRUCTION I_3 I_2 I_1 I_0	FUNCTION	TEST INPUT	NEXT ADDR SOURCE	FILE	COUNTER	MAP-E	PL-E
JZ	L L L L	JUMP ZERO	X	D	HOLD	L L	H	L
CJS	L L L H	COND JSB PL	L	PC	HOLD	HOLD	H	L
			H	D	PUSH	HOLD	H	L
JMAP	L L H L	JUMP MAP	X	D	HOLD	HOLD	L	H
CJP	L L H H	COND JUMP PL	L	PC	HOLD	HOLD	H	L
			H	D	HOLD	HOLD	H	L
PUSH	L H L L	PUSH/COND LD CNTR	L	PC	PUSH	HOLD	H	L
			H	PC	PUSH	LOAD	H	L
JSRP	L H L H	COND JSB R/PL	L	R	PUSH	HOLD	H	L
			H	D	PUSH	HOLD	H	L
CJV	L H H L	COND JUMP VECTOR	L	PC	HOLD	HOLD	H	H
			H	D	HOLD	HOLD	H	H
JRP	L H H H	COND JUMP R/PL	L	R	HOLD	HOLD	H	L
			H	D	HOLD	HOLD	H	L
RFCT	H L L L	REPEAT LOOP, CNTR ≠ 0	L	F	HOLD	DEC	H	L
			H	PC	POP	HOLD	H	L
RPCT	H L L H	REPEAT PL, CNTR ≠ 0	L	D	HOLD	DEC	H	L
			H	PC	HOLD	HOLD	H	L
CRTN	H L H L	COND RTN	L	PC	HOLD	HOLD	H	L
			H	F	POP	HOLD	H	L
CJPP	H L H H	COND JUMP PL & POP	L	PC	HOLD	HOLD	H	L
			H	D	POP	HOLD	H	L
LDCT	H H L L	LOAD CNTR & CONTINUE	X	PC	HOLD	LOAD	H	L
LOOP	H H L H	TEST END LOOP	L	F	HOLD	HOLD	H	L
			H	PC	POP	HOLD	H	L
CONT	H H H L	CONTINUE	X	PC	HOLD	HOLD	H	L
JP	H H H H	JUMP PL	X	D	HOLD	HOLD	H	L

microinstruction word would be formed, similar to the 12-bit word formed using the 2910. The mapping PROM and pipeline register are identical with the previous example, so we can therefore turn to the new features. The 25LS163s form an event counter, and a multiplexer (23LS153) and a D-type flip-flop (74S174) are used for handling the output from the pipeline register.

Finally, it might be asked how the computer control unit interfaces with the slice microprocessors to form an actual microcomputer. Figure 7.11 presents such an example of a 16-bit microcomputer constructed using the 2910 Computer Control Unit, together with the 2901 microprocessor.

Instruction Set

To illustrate the operation of the computer control unit under various instructions of the 2910 or 29811, we consider each instruction and the effect it has on the execution of microinstructions in the microprogram memory.

Figure 7.12 is a representation of instruction flow in the CCU for each of the 16 instructions indicated by mnemonics in Tables 7.1, 7.3, and 7.4. The designation of specific instruction numbers (e.g., 50 and 51) is purely arbitrary; the point is to show instruction flow. Each dot represents a different microinstruction in the pipeline register. "CONT" represents the CONTINUE instruction, that is, the microprogram memory address is simply incremented by one.

*Reprinted with permission of Advanced Micro Devices.

Since the explanation of the instruction set of the 2910 and 29811 is relatively straightforward, the explanation beginning on page 179 is taken directly from the product literature of AMD (for Table 1, see Table 7.1 in this book; for Figure 4, see our Figure 7.12).*

Software

Microprogram software is also important in the development of a 2901 system. Just as development system software exists for other microprocessors, there are development programs for microprogrammable systems as well. One such program is the AMDASM microprogram assembler for use with the 2901 system.

The AMDASM program produces microprogram instructions for execution by the user's specific hardware system. The program therefore permits the user to define various registers and hardware logic structures. The user then codes the specific program with symbolic source statements using the defined terms. The AMDASM then operates on the source statements to generate a binary output, together with a program listing, much like the operation of an ordinary assembler.

References

Am2903 Bipolar Microprocessor Slice, Am2910 Microprogram Controller, Product Specification, Advanced Micro Devices, Sunnyvale, Calif., 1978.

Build a Microcomputer, Advanced Micro Devices, Sunnyvale, Calif., 1978.

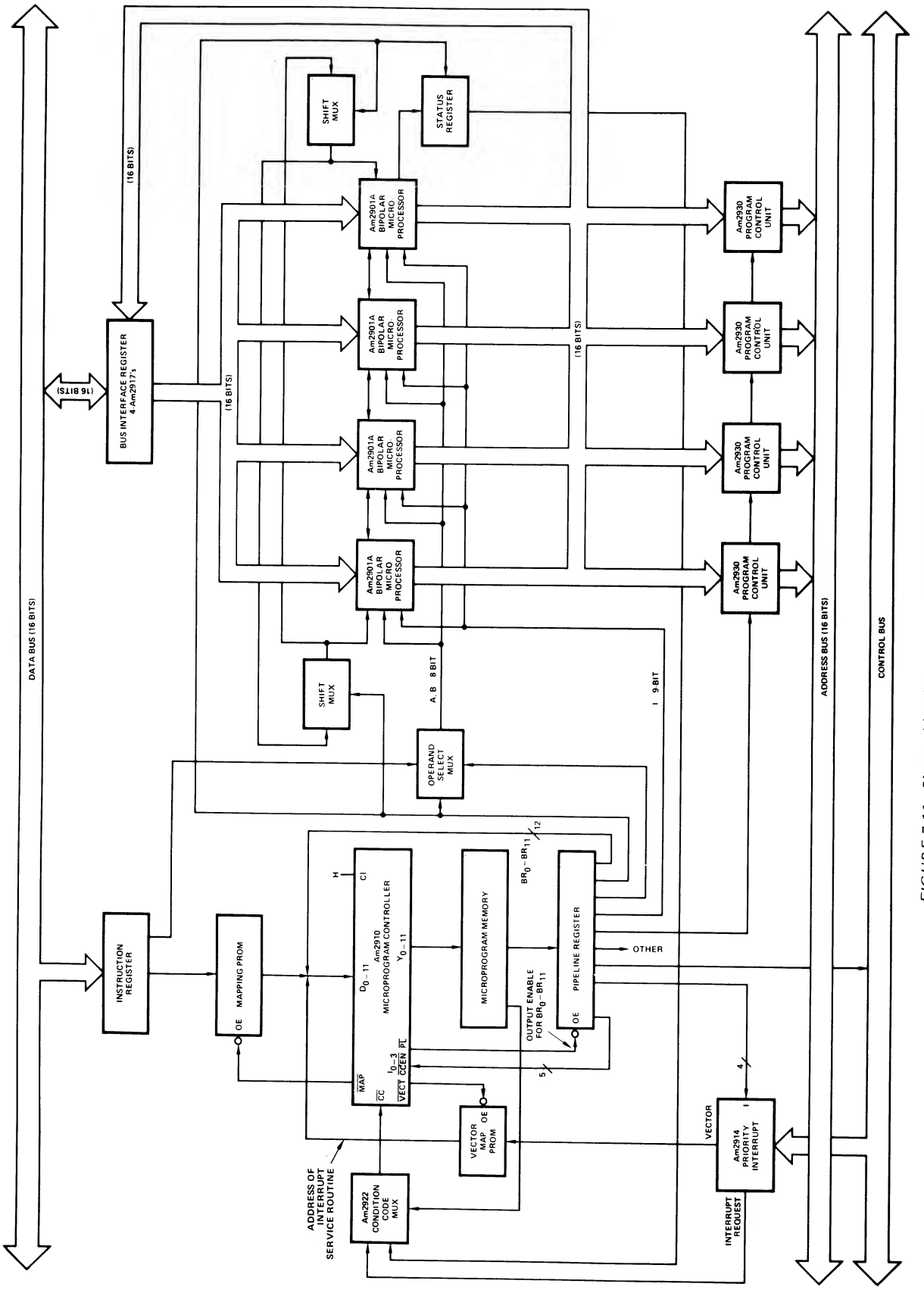


FIGURE 7.11 Sixteen-bit microcomputer using AMD 2910 and 2901.

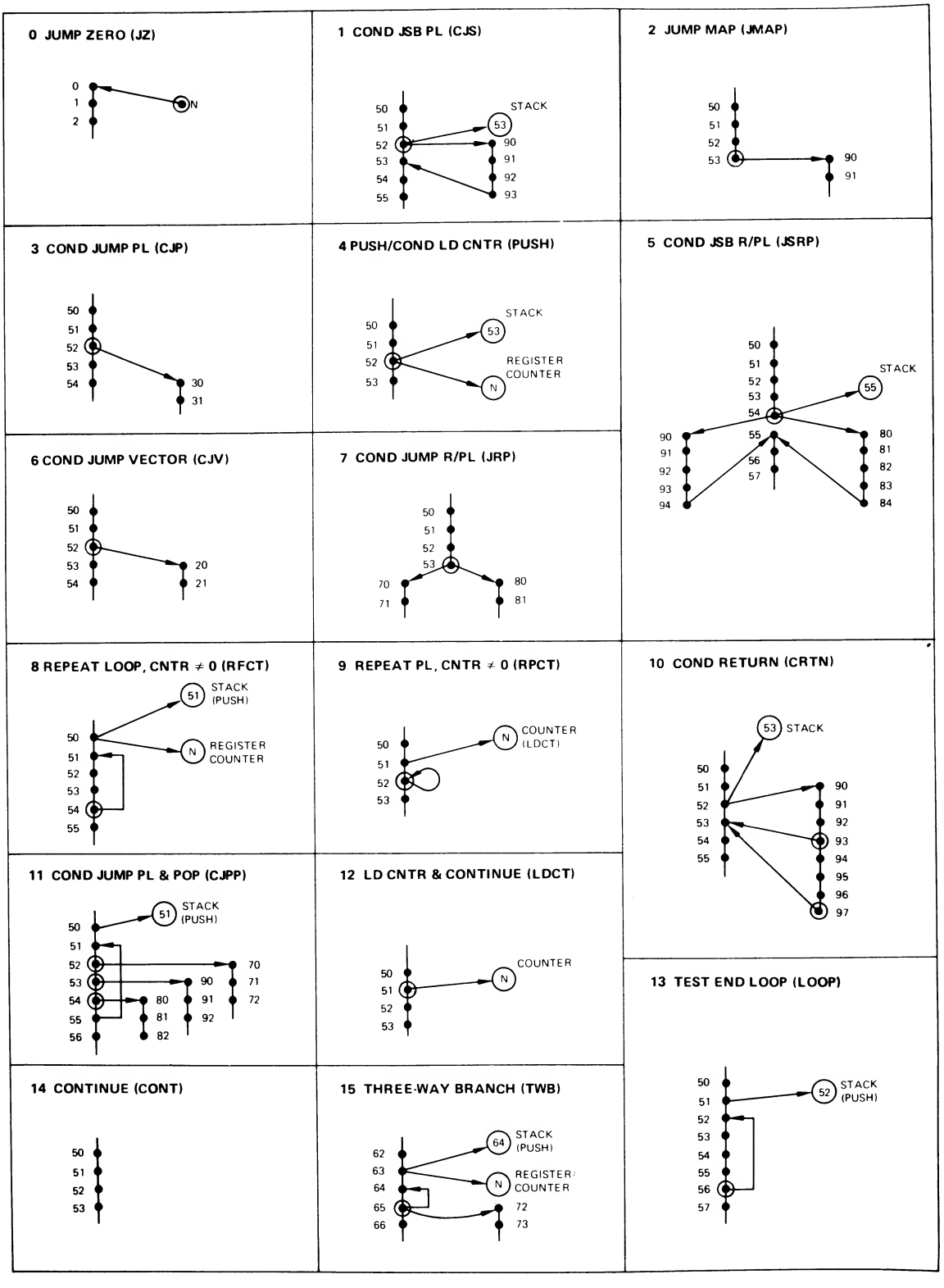


FIGURE 7.12 Instruction flow in the AMD 2910.

THE Am2910 INSTRUCTION SET

The Am2910 provides 16 instructions which select the address of the next microinstruction to be executed. Four of the instructions are unconditional – their effect depends only on the instruction. Ten of the instructions have an effect which is partially controlled by an external, data-dependent condition. Three of the instructions have an effect which is partially controlled by the contents of the internal register/counter. The instruction set is shown in Table I. In this discussion it is assumed that C_n is tied HIGH.

In the ten conditional instructions, the result of the data-dependent test is applied to $\overline{CC}$. If the $\overline{CC}$ input is LOW, the test is considered to have been passed, and the action specified in the name occurs; otherwise, the test has failed and an alternate (often simply the execution of the next sequential microinstruction) occurs. Testing of $\overline{CC}$ may be disabled for a specific microinstruction by setting $\overline{CCEN}$ HIGH, which unconditionally forces the action specified in the name; that is, it forces a pass. Other ways of using $\overline{CCEN}$ include (1) tying it HIGH, which is useful if no microinstruction is data-dependent; (2) tying it LOW if data-dependent instructions are never forced unconditionally; or (3) tying it to the source of Am2910 instruction bit I_0, which leaves instructions 4, 6, and 10 as data-dependent but makes others unconditional. All of these tricks save one bit of microcode width.

The effect of three instructions depends on the contents of the register/counter. Unless the counter holds a value of zero, it is decremented; if it does hold zero, it is held and a different microprogram next address is selected. These instructions are useful for executing a microinstruction loop a known number of times. Instruction 15 is affected both by the external condition code and the internal register/counter.

Perhaps the best technique for understanding the Am2910 is to simply take each instruction and review its operation. In order to provide some feel for the actual execution of these instructions, Figure 4 is included and depicts examples of all 16 instructions.

The examples given in Figure 4 should be interpreted in the following manner: The intent is to show microprogram flow as various microprogram memory words are executed. For example, the CONTINUE instruction, instruction number 14, as shown in Figure 4, simply means that the contents of microprogram memory word 50 is executed, then the contents of word 51 is executed. This is followed by the contents of microprogram memory word 52 and the contents of microprogram memory word 53. The microprogram addresses used in the examples were arbitrarily chosen and have no meaning other than to show instruction flow. The exception to this is the first example, JUMP ZERO, which forces the microprogram location counter to address ZERO. Each dot refers to the time that the contents of the microprogram memory word is in the pipeline register. While no special symbology is used for the conditional instructions, the test to follow will explain what the conditional choices are in each example.

It might be appropriate at this time to mention that AMD has a microprogram assembler called AMDASM, which has the capability of using the Am2910 instructions in symbolic representation. AMDASM's Am2910 instruction symbolics (or mnemonics) are given in Figure 4 for each instruction and are also shown in Table I.

Instruction 0, JZ (JUMP and ZERO, or RESET) unconditionally specifies that the address of the next microinstruction is zero. Many designs use this feature for power-up sequences and provide the power-up firmware beginning at microprogram memory word location 0.

Instruction 1 is a CONDITIONAL JUMP-TO-SUBROUTINE via the address provided in the pipeline register. As shown in Figure 4, the machine might have executed words at address 50, 51, and 52. When the contents of address 52 is in the pipeline register, the next address control function is the CONDITIONAL JUMP-TO-SUBROUTINE. Here, if the test is passed, the next instruction executed will be the contents of microprogram memory location 90. If the test has failed, the JUMP-TO-SUBROUTINE will not be executed; the contents of microprogram memory location 53 will be executed instead. Thus, the CONDITIONAL JUMP-TO-SUBROUTINE instruction at location 52 will cause the instruction either in location 90 or in location 53 to be executed next. If the TEST input is such that location 90 is selected, value 53 will be pushed onto the internal stack. This provides the return linkage for the machine when the subroutine beginning at location 90 is completed. In this example, the subroutine was completed at location 93 and a RETURN-FROM-SUBROUTINE would be found at location 93.

Instruction 2 is the JUMP MAP instruction. This is an unconditional instruction which causes the $\overline{MAP}$ output to be enabled so that the next microinstruction location is determined by the address supplied via the mapping PROMs. Normally, the JUMP MAP instruction is used at the end of the instruction fetch sequence for the machine. In the example of Figure 4, microinstructions at locations 50, 51, 52, and 53 might have been the fetch sequence and at its completion at location 53, the jump map function would be contained in the pipeline register. This example shows the mapping PROM outputs to be 90; therefore, an unconditional jump to microprogram memory address 90 is performed.

Instruction 3, CONDITIONAL JUMP PIPELINE, derives its branch address from the pipeline register branch address value (BR_0 – BR_{11} in Figure 2). This instruction provides a technique for branching to various microprogram sequences depending upon the test condition inputs. Quite often, state machines are designed which simply execute tests on various inputs waiting for the condition to come true. When the true condition is reached, the machine then branches and executes a set of microinstructions to perform some function. This usually has the effect of resetting the input being tested until some point in the future. Figure 4 shows the conditional jump via the pipeline register address at location 52. When the contents of microprogram memory word 52 are in the pipeline register, the next address will be either location 53 or location 30 in this example. If the test is passed, the value currently in the pipeline register (3) will be selected. If the test fails, the next address selected will be contained in the microprogram counter which, in this example, is 53.

Instruction 4 is the PUSH/CONDITIONAL LOAD COUNTER instruction and is used primarily for setting up loops in microprogram firmware. In Figure 4, when instruction 52 is in the pipeline register, a PUSH will be made onto the stack and the counter will be loaded based on the condition. When a PUSH occurs, the value pushed is always the next sequential instruction address. In this case, the address is 53. If the test fails, the counter is not loaded; if it is passed, the counter is loaded with the value contained in the pipeline register branch address field. Thus, a single microinstruction can be used to set up a loop to be executed a specific number of times. Instruction 8 will

describe how to use the pushed value and the register/counter for looping.

Instruction 5 is a CONDITIONAL JUMP-TO-SUBROUTINE via the register/counter or the contents of the PIPELINE register. As shown in Figure 4, a PUSH is always performed and one of two subroutines executed. In this example, either the subroutine beginning at address 80 or the subroutine beginning at address 90 will be performed. A return-from-subroutine (instruction number 10) returns the microprogram flow to address 55. In order for this microinstruction control sequence to operate correctly, both the next address fields of instruction 53 and the next address fields of instruction 54 would have to contain the proper value. Let's assume that the branch address fields of instruction 53 contain the value 90 so that it will be in the Am2910 register/counter when the contents of address 54 are in the pipeline register. This requires that instruction at address 53 load the register/counter. Now, during the execution of instruction 5 (at address 54), if the test failed, the contents of the register (value = 90) will select the address of the next microinstruction. If the test input passes, the pipeline register contents (value = 80) will determine the address of the next microinstruction. Therefore, this instruction provides the ability to select one of two subroutines to be executed based on a test condition.

Instruction 6 is a CONDITIONAL JUMP VECTOR instruction which provides the capability to take the branch address from a third source heretofore not discussed. In order for this instruction to be useful, the Am2910 output, $\overline{\text{VECT}}$ is used to control a three-state control input of a register, buffer, or PROM containing the next microprogram address. This instruction provides one technique for performing interrupt type branching at the microprogram level. Since this instruction is conditional, a pass causes the next address to be taken from the vector source, while failure causes the next address to be taken from the microprogram counter. In the example of Figure 4, if the CONDITIONAL JUMP VECTOR instruction is contained at location 52, execution will continue at vector address 20 if the TEST input is HIGH and the microinstruction at address 53 will be executed if the TEST input is LOW.

Instruction 7 is a CONDITIONAL JUMP via the contents of the Am2910 REGISTER/COUNTER or the contents of the PIPELINE register. This instruction is very similar to instruction 5; the conditional jump-to-subroutine via R or PL. The major difference between instruction 5 and instruction 7 is that no push onto the stack is performed with 7. Figure 4 depicts this instruction as a branch to one of two locations depending on the test condition. The example assumes the pipeline register contains the value 70 when the contents of address 52 is being executed. As the contents of address 53 is clocked into the pipeline register, the value 70 is loaded into the register/counter in the Am2910. The value 80 is available when the contents of address 53 is in the pipeline register. Thus, control is transferred to either address 70 or address 80 depending on the test condition.

Instruction 8 is the REPEAT LOOP, COUNTER $\neq$ ZERO instruction. This microinstruction makes use of the decrementing capability of the register/counter. To be useful, some previous instruction, such as 4, must have loaded a count value into the register/counter. This instruction checks to see whether the register/counter contains a non-zero value. If so, the register/counter is decremented, and the address of the next microinstruction is taken from the top of the stack. If the register counter contains zero, the loop exit condition is occuring; control falls through to the next sequential microinstruction by selecting uPC; the stack is POP'd by decrementing the stack pointer, but the contents of the top of the stack are thrown away.

An example of the REPEAT LOOP, COUNTER $\neq$ ZERO instruction is shown in Figure 4. In this example, location 50 most likely would contain a PUSH/CONDITIONAL LOAD COUNTER instruction which would have caused address 51 to be PUSHed on the stack and the counter to be loaded with the proper value for looping the desired number of times.

In this example, since the loop test is made at the end of the instructions to be repeated (microaddress 54), the proper value to be loaded by the instructions at address 50 is one less than the desired number of passes through the loop. This method allows a loop to be executed 1 to 4096 times. If it is desired to execute the loop from 0 to 4095 times, the firmware should be written to make the loop exit test immediately after loop entry.

Single-microinstruction loops provide a highly efficient capability for executing a specific microinstruction a fixed number of times. Examples include fixed rotates, byte swap, fixed point multiply, and fixed point divide.

Instruction 9 is the REPEAT PIPELINE REGISTER, COUNTER $\neq$ ZERO instruction. This instruction is similar to instruction 8 except that the branch address now comes from the pipeline register rather than the file. In some cases, this instruction may be thought of as a one-word file extension; that is, by using this instruction, a loop with the counter can still be performed when subroutines are nested five deep. This instruction's operation is very similar to that of instruction 8. The differences are that on this instruction, a failed test condition causes the source of the next microinstruction address to be the D inputs; and, when the test condition is passed, this instruction does not perform a POP because the stack is not being used.

In the example of Figure 4, the REPEAT PIPELINE, COUNTER $\neq$ ZERO instruction is instruction 52 and is shown as a single microinstruction loop. The address in the pipeline register would be 52. Instruction 51 in this example could be the LOAD COUNTER AND CONTINUE instruction (number 12). While the example shows a single microinstruction loop, by simply changing the address in a pipeline register, multi-instruction loops can be performed in this manner for a fixed number of times as determined by the counter.

Instruction 10 is the conditional RETURN-FROM-SUBROUTINE instruction. As the name implies, this instruction is used to branch from the subroutine back to the next microinstruction address following the subroutine call. Since this instruction is conditional, the return is performed only if the test is passed. If the test is failed, the next sequential microinstruction is performed. The example in Figure 4 depicts the use of the conditional RETURN-FROM-SUBROUTINE instruction in both the conditional and the unconditional modes. This example first shows a jump-to-subroutine at instruction location 52 where control is transferred to location 90. At location 93, a conditional RETURN-FROM-SUBROUTINE instruction is performed. If the test is passed, the stack is accessed and the program will transfer to the next instruction at address 53. If the test is failed, the next microinstruction at address 94 will be executed. The program will continue to address 97 where the subroutine is complete. To perform an unconditional RETURN-FROM-SUBROUTINE, the conditional RETURN-FROM-SUBROUTINE instruction is executed unconditionally; the microinstruction at address 97 is programmed to force

$\overline{CCEN}$ HIGH, disabling the test and the forced PASS causes an unconditional return.

Instruction 11 is the CONDITIONAL JUMP PIPELINE register address and POP stack instruction. This instruction provides another technique for loop termination and stack maintenance. The example in Figure 4 shows a loop being performed from address 55 back to address 51. The instructions at locations 52, 53, and 54 are all conditional JUMP and POP instructions. At address 52, if the TEST input is passed, a branch will be made to address 70 and the stack will be properly maintained via a POP. Should the test fail, the instruction at location 53 (the next sequential instruction) will be executed. Likewise, at address 53, either the instruction at 90 or 54 will be subsequently executed, respective to the test being passed or failed. The instruction at 54 follows the same rules, going to either 80 or 55. An instruction sequence as described here, using the CONDITIONAL JUMP PIPELINE and POP instruction, is very useful when several inputs are being tested and the microprogram is looping waiting for any of the inputs being tested to occur before proceeding to another sequence of instructions. This provides the powerful jump-table programming technique at the firmware level.

Instruction 12 is the LOAD COUNTER AND CONTINUE instruction, which simply enables the counter to be loaded with the value at its parallel inputs. These inputs are normally connected to the pipeline branch address field which (in the architecture being described here) serves to supply either a branch address or a counter value depending upon the microinstruction being executed. There are altogether three ways of loading the counter — the explicit load by this instruction 12; the conditional load included as part of instruction 4; and the use of the $\overline{RLD}$ input along with any instruction. The use of $\overline{RLD}$ with any instruction overrides any counting or decrementation specified in the instruction, calling for a load instead. Its use provides additional microinstruction power, at the expense of one bit of microinstruction width. This instruction 12 is exactly equivalent to the combination of instruction 14 and $\overline{RLD}$ LOW. Its purpose is to provide a simple capability to load the register/counter in those implementations which do not provide microprogrammed control for $\overline{RLD}$.

Instruction 13 is the TEST END-OF-LOOP instruction, which provides the capability of conditionally exiting a loop at the bottom; that is, this is a conditional instruction that will cause the microprogram to loop, via the file, if the test is failed else to continue to the next sequential instruction. The example in Figure 4 shows the TEST END-OF-LOOP microinstruction at address 56. If the test fails, the microprogram will branch to address 52. Address 52 is on the stack because a PUSH instruction had been executed at address 51. If the test is passed at instruction 56, the loop is terminated and the next sequential microinstruction at address 57 is executed, which also causes the stack to be POP'd; thus, accomplishing the required stack maintenance.

Instruction 14 is the CONTINUE instruction, which simply causes the microprogram counter to increment so that the next sequential microinstruction is executed. This is the simplest microinstruction of all and should be the default instruction which the firmware requests whenever there is nothing better to do.

Instruction 15, THREE-WAY BRANCH, is the most complex. It provides for testing of both a data-dependent condition and the counter during one microinstruction and provides for selecting among one of three microinstruction addresses as the next microinstruction to be performed. Like instruction 8, a previous instruction will have loaded a count into the register/counter while pushing a microbranch address onto the stack. Instruction 15 performs a decrement-and-branch-until-zero function similar to instruction 8. The next address is taken from the top of the stack until the count reaches zero; then the next address comes from the pipeline register. The above action continues as long as the test condition fails. If at any execution of instruction 15 the test condition is passed, no branch is taken; the microprogram counter register furnishes the next address. When the loop is ended, either by the count becoming zero, or by passing the conditional test, the stack is POP'd by decrementing the stack pointer, since interest in the value contained at the top of the stack is then complete.

The application of instruction 15 can enhance performance of a variety of machine-level instructions. For instance, (1) a memory search instruction to be terminated either by finding a desired memory content or by reaching the search limit; (2) variable-field-length arithmetic terminated early upon finding that the content of the portion of the field still unprocessed is all zeroes; (3) key search in a disc controller processing variable length records; (4) normalization of a floating point number.

As one example, consider the case of a memory search instruction. As shown in Figure 4, the instruction at microprogram address 63 can be Instruction 4 (PUSH), which will push the value 64 onto the microprogram stack and load the number N, which is one less than the number of memory locations to be searched before giving up. Location 64 contains a microinstruction which fetches the next operand from the memory area to be searched and compares it with the search key. Location 65 contains a microinstruction which tests the result of the comparison and also is a THREE-WAY BRANCH for microprogram control. If no match is found, the test fails and the microprogram goes back to location 64 for the next operand address. When the count becomes zero, the microprogram branches to location 72, which does whatever is necessary if no match is found. If a match occurs on any execution of the THREE-WAY BRANCH at location 65, control falls through to location 66 which handles this case. Whether the instruction ends by finding a match or not, the stack will have been POP'd once, removing the value 64 from the top of the stack.

8

interfacing

Interfacing refers to the connection of a microprocessor or microcomputer to some external or peripheral device. Several different levels of interfacing must be considered.

One can speak of interfacing on several different levels: mechanical, electrical, functional (computer to application), and user (application to application). The concept of system compatibility generally means compatibility on all of these levels. If the devices or subsystems originate from the same manufacturer and are designed and intended to operate together, then the task of the user is simply to connect the units with the supplied cables and begin operation. On the other hand, if the user has devices from different manufacturers then interfacing becomes a significant problem.

The mechanical or physical interface level refers to the mechanical characteristics of the connection means between two devices or subsystems. Such connection means may take the form of a connector plug having a certain size and shape with a given number of pins. Other connection means may simply be the pins on the edge of a printed circuit board, with the number of pins and pin-to-pin spacing being specified.

The electrical interface level refers to the definition of the electrical signals on each of the pins of the connection means. Such a definition includes the voltage level and the specific function of the electrical signal on the pins (e.g., 0 V represents a binary 0, 5 V represents a binary 1, and the pin represents a data bit).

The functional interface level refers to the rules for sending signals over the connection means, including the specific sequence of signals generated by both devices that defines the purpose of the communication. Such a signal exchange between two devices is known as "handshaking," and the rules for such a signal interchange are sometimes referred to as a protocol.

Finally, the user interface level refers to the meaning of the data or information being conveyed from one device to another. The meaning of data is usually defined within the context of a given operating system or application program: a bit sequence or character string can therefore be interpreted as "turn motor on" or "valve is closed."

These four interface levels are nested sets, with mechanical compatibility being the largest set. It is possible to imagine two devices being mechanically compatible (i.e., they have the same plug or connector), but completely incompatible on electrical, functional, or user grounds. Similarly, two devices may be fully compatible on mechanical, electrical, and functional grounds, but no useful data interchange can take place because they are not compatible on the user interface level. A simple example of this latter situation is when one user is

using USASCII code and the other is using EBCDIC code.

In the present chapter, the following peripheral devices and interfacing issues are considered:

- Memory interfacing.
- I/O interface control.
- CRT interfacing.
- Serial communications device interfacing.
- Multiprotocol data link control interfacing.
- Encrypted data link interfacing.
- Floating point processors.

Memory Interfacing

Memory interfacing is concerned with interfacing the microprocessor to memory devices. Memory devices fall into two broad categories.

- Random access memories.
- Mass memories.

There are basically two types of random access memories: read/write random access memories (RAM), and read-only random access memories (ROM). RAM is used for general-purpose storage in a computer system, with data or programs being read or written into specified memory locations. ROM is used for the relatively permanent storage of instructions or data.

Mass memories are those that have larger storage capacities and that by design are not "random access." Mass memories include:

- Magnetic tape storage.
- Magnetic disk storage.
- Solid state mass storage: CCDs and bubbles.
- Electrooptical memories.

Modern microprocessor system design is not only concerned with the technical details of interfacing to each of these types of memory devices, but also the basic capabilities and limitations of each of these memory technologies. Figure 8.1 places memory technology in perspective by considering the various storage capacities and access times of each device.

Another important comparison is to include the cost per bit as a parameter. Figure 8.2 is a plot of the typical OEM price (in cents/bit) versus the access time for the major memory technologies.

In selecting memory for use in a system the designer is concerned with the following parameters:

- Total memory size.
- Price per bit.
- Access time (including access time to each block, and the time to transfer a word in a block).
- Power requirements.
- Volatility.
- Reliability and repairability.
- Maturity of technology.

As Figures 8.1 and 8.2 indicate, different memory technologies have different characteristics. Although there are some overlapping areas (for example, floppy versus fixed disks), the specific nature of the application generally favors one memory technology over another. One should also note some prominent gaps in comparing technologies, such as between MOS RAMs and serial memories such as CCDs and bubble devices, and the

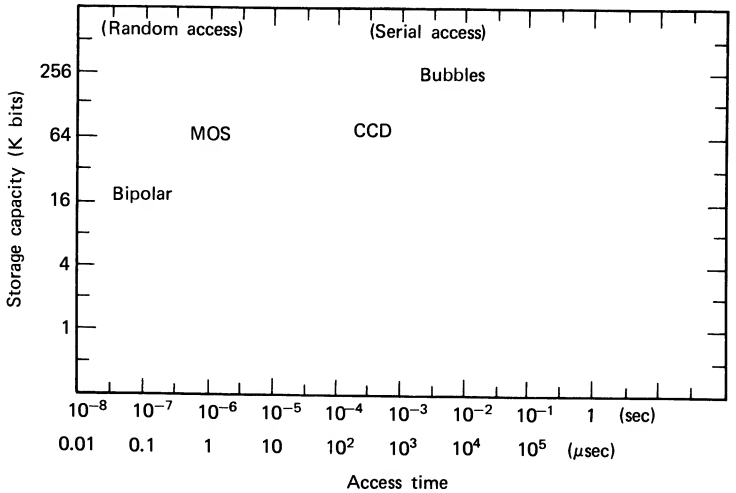


FIGURE 8.1 Memory technology: storage capacity versus access time.

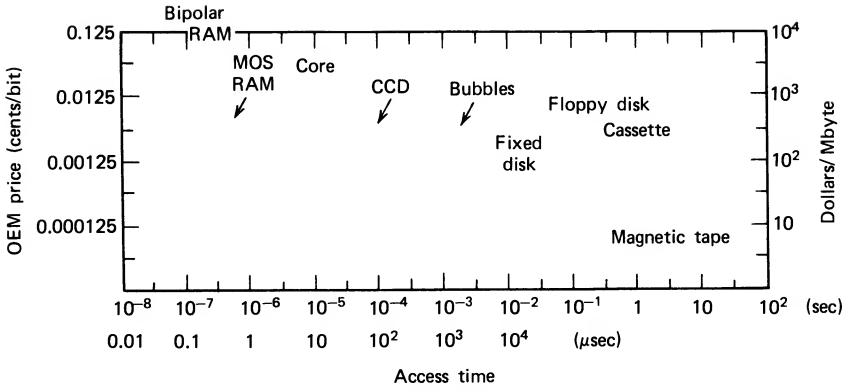


FIGURE 8.2 Memory technology: OEM price versus access time.

general technological trends indicated by small arrows in Figures 8.1 and 8.2.

RAMs

There are several aspects of RAMs to consider in the present survey:

- Technology.
- Configuration.
- Interfacing.

Technology. The technology of random access memory devices refers to both the semiconductor technology and the circuit technology. RAM may be designed with any type of semiconductor technology—MOS, bipolar, and so on. The semiconductor technology chosen for the RAM is generally based on the semiconductor technology of the other major components of the system, such as the processor, in order to maintain compatibility both in terms of timing and electrical interfacing.

There are two basic types of circuit technology for semiconductor RAMs: static and dynamic.

"A static memory circuit is one that stores data in a flip-flop circuit and thus is able to maintain that data as long as power is applied to the memory circuit. A dynamic memory circuit is one that stores data in a capacitor in a circuit and thus must be periodically refreshed in order to maintain that data.

Static RAM has the advantage of requiring simpler external drive circuitry and for that reason is most frequently used in computing systems. Dynamic RAM is denser and less expensive, and has lower power dissipation than static RAM, but such advantages are only important for larger memory arrays, such as in mainframe computers.

"Dynamic RAM also has more critical timing requirements than static RAM. Some central processor boards will not operate with dynamic memory boards due to slight differences in memory access synchronization signals."*

A brief remark on terminology in designating memory devices would be helpful here. In considering semiconductor memories, reference is usually made to the storage capacity of the memory in *bits*, that is, 1K, 12K, 64K, and so on. It must be realized that the K in these designations refers to 1024. Thus a 1K chip is a memory device that stores 1024 bits. It is irrelevant how the bits are configured: 1024 X 1 or 256 X 4 still refer to a 1K device.

It must be noted that a 64K device stores 2^{16} or 65,536 bits. The use of a capital K to represent 1024 is fairly consistent in the industry, although it must be pointed out that certain short-sighted would-be manufacturers of such a device have at times referred to such components as 65K.

For completeness, it should also be pointed out that a small k is occasionally also used in place of K to represent 1024 for typographical reasons. Such usage is, however, less common that that of the capital K. For those who use both capital K and small k, the capital K refers to 1024, and the small k to 1000 (as in "kHz" for kilohertz). In such cases however, the small k is not used to refer to the storage capacity of a memory device.

It must be emphasized that the storage capacity of a memory *device* is measured in *bits*, but the storage capacity of a memory *system* or board in a computer is measured in bytes, or words. Thus a 16K memory in a computer refers to 16K bytes.

It would also be worthwhile to note how memory addresses are designated in random access memories. The hexadecimal representation is used to represent the address of successive memory bytes. The first memory byte begins with address 0000, and successive bytes are

*Adapted from Daniel R. McGlynn, *Personal Computing* (Wiley, New York, 1979).

designated 0001, 0002, and so on. In a 16K memory block the last byte then has the address 3FFF. In a 64K memory block the last address is FFFF, as shown in Figure 8.3. It should be emphasized that byte-addressing of memories is used even with 16-bit microprocessors that are capable of both byte and word addressing. Figure 8.3 shows how a 64K block may be broken up into subblocks of 16K each, and the addresses of the beginning and ending byte of each subblock.

For a 16-bit microprocessor with a 1-megabyte addressing capability, the last byte has the address FFFFF. For microprocessors with an addressing capability greater than 1 megabyte, the first byte of the second megabyte block begins with address 100000 and ends with address 1FFFFF. The third megabyte block begins with 200000 and ends with 2FFFFF, and so on.

Some historical perspective would also be useful here. The first semiconductor random access memories were 256 X 1 bit static devices and were generally used for buffer storage applications. The Intel 1101, introduced in 1969 is typical of such devices. In 1971 Intel introduced the 1103, a 1024 X 1 bit dynamic MOS RAM. The 1103 proved to be one of the most popular integrated circuits ever introduced and demonstrated that semiconductor memory could be competitive with core memory.

The 4K devices were the next generation of semiconductor memory. Again using Intel as an example, the 2107 4K dynamic RAM was introduced in 1973 in a 22-pin package. There was considerable controversy in design engineering circles in the mid-1970s concerning the optimum number of pins in a 4K RAM package; some of these considerations are discussed in the next section on memory configurations. Sixteen-pin 4K dynamic RAMs began taking the lead in 1977 and in some circles may even be considered "standard" by now.

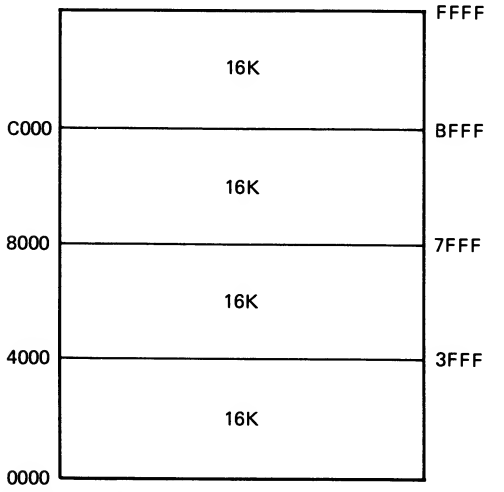


FIGURE 8.3 Memory addresses in hexadecimal representation.

In 1977 the next generation of memory devices—16K dynamic RAMs—was introduced. The 16-pin Mostek 4116 is one of the leading design examples of integrated circuits in this category and is considered in greater detail later in this chapter.

Finally, the fourth generation of semiconductor memory devices—the 64K dynamic RAMs—were announced in late 1978. The Texas Instruments 64K 4164 is one of the first contenders in the new generation of devices.

The progress in semiconductor random access memories may be illustrated by considering the improvement in access time. Figure 8.4 shows the access time of various members of Intel's family of static RAMs, indicating an improvement from 500 to nsec in the 2102 in 1972 to 20 nsec in the 2125H in 1979.

It is instructive to point out some of the MOS circuit design changes made in the last few years that have been principally responsible for the advances in speed and density of such circuits, particularly in dynamic RAMs.

Some of the earlier designs for MOS dynamic RAMs included three MOS field effect transistors in each memory cell, such as shown in Figures 8.5*a* and 8.5*b*. Such a configuration had drawbacks, such as a relatively long bit line due to the large number of elements in each cell. One consequence of the length of the bit line is the capacitance associated with the bit line, which is correspondingly large. Such capacitance reduces the speed of the memory, and therefore the use of such four-transistor cells in RAMs was disadvantageous and soon lost popularity as newer circuit designs evolved.

The disadvantage of the cell in Figure 8.5*a* is that the amplitude of the signal applied to the select line during the read cycle must be carefully controlled. As indicated in the figure, the cell is coupled to a select line, a write data line, a read data line, and a common line. The capacitor shown in dotted lines is typically the parasitic capacitance associated with the MOS devices to which it is connected. To write information in the cell, a signal is applied to the select line that allows a signal (if one is applied) on the write data line to flow from the write data line to the capacitor, where the information is stored in the form of a charge. To read information from the cell, both the read data line and the write data line are precharged. A signal is then applied to the select line, causing the MOS device in the upper right hand portion of the cell to discharge the read data line via the common line if the MOS device in the lower right hand corner is conducting. This MOS device is conducting only if a charge has previously been stored on the capacitor.

The improved three-transistor cell shown in Figure 8.5*b* includes separate select write lines and select read lines, in addition to the write data line and read data line.

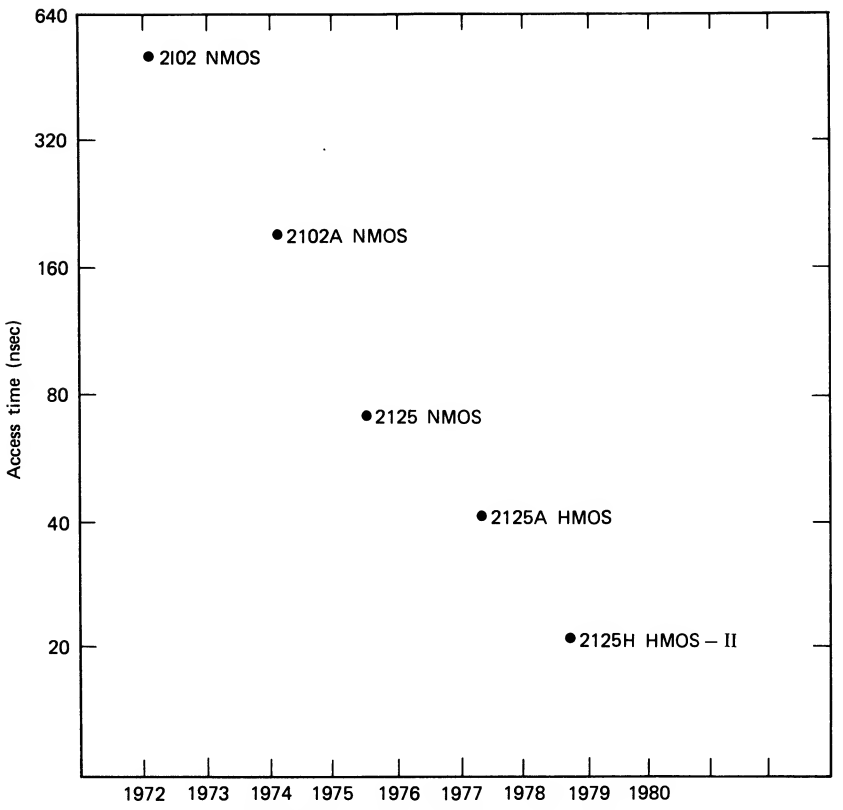


FIGURE 8.4 Static memory access times.

The cell shown in Figure 8.5*b* operates as follows. Data are written into the cell by placing a high signal on the select write line and by applying the data to be written into the cell on the write data line. To read information from the cell, the read data line is precharged to a high level and a high level is also applied to the select read line. The select write line then senses the inverse of the data stored on the capacitor. If a one bit is stored on the capacitor, the charge on the select write line discharges through the ground or common line since the two MOS devices in the right hand portion of the figure will conduct. If a zero bit or no charge exists on the capacitor, then the precharge remains on the select write line and can be detected by external circuitry.

Many of the newer circuit designs utilize only one transistor for each memory cell as can be described with reference to Figure 8.6.

Figure 8.6 is a schematic diagram of an example of a one transistor per bit random access memory. The two vertical lines are digit lines that provide inputs to a differential sense amplifier. These digit lines are connected to two MOS field effect transistors that form a precharge circuit. They are connected through the precharging MOS field effect transistors to the source of potential V_{DD}. To the gates of the field effect transistors in the precharge circuit is applied the precharge signal P. At the opposite end of the digit lines is a transfer circuit consisting of two transfer MOS field effect transistors that are turned on by a signal on their gates that is connected to a Y decoder. When a specific memory column or Y value is selected by the decoder, the gates of the transfer FETs are activated and the digit lines then have access to the input or output bus, shown as the two parallel lines at the bottom of Figure 8.6.

The differential sense amplifier is associated with a pair of bit lines that connected to the differential inputs of each sense amplifier. The signals sent out on this pair of bit lines are true and complement signals. These true and complement signals are in turn transmitted to a pair of input or output bus lines, which are in turn applied to the inputs of a main differential amplifier.

The two parallel lines shown in the upper half of Figure 8.6 are word lines and are arranged at right angles to the bit lines. The word lines are connected to an X decoder so that a signal is supplied to the appropriate word line when the decoder selects a specific memory row.

The typical memory cell is shown as a single field

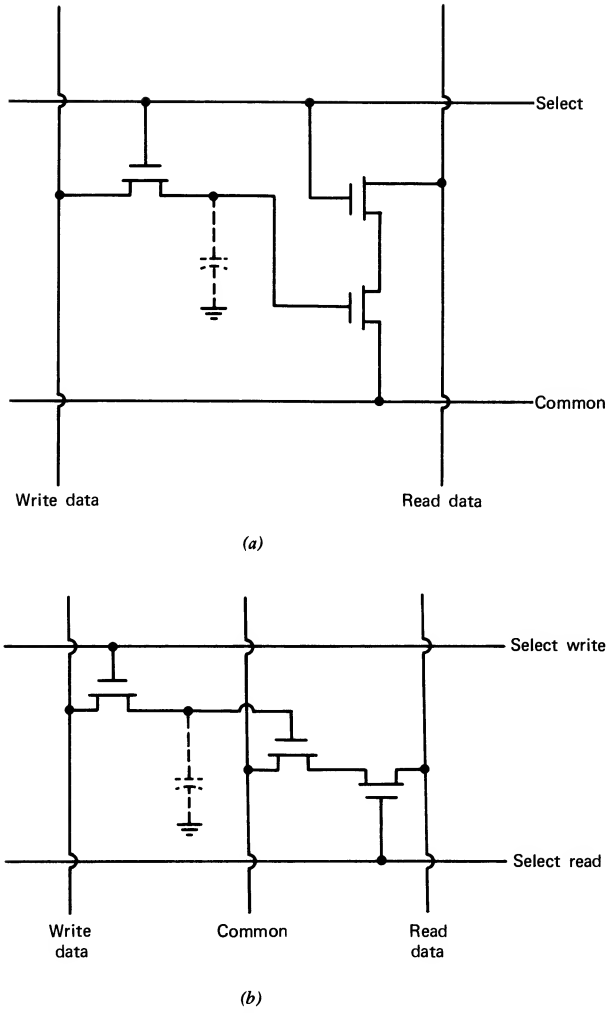


FIGURE 8.5 Three-transistor dynamic memory cell.

effect transistor having its gate connected to a word line and one of the conduction path electrodes connected to one of the bit lines.

It is important to understand how the memory circuit shown in Figure 8.6 works. When a precharge signal P is applied to the gate of the precharge field effect transistors they conduct and the bit lines are precharged to the voltage level V_{DD}. The I/O points of the sense amplifier are also precharged to this level. Assume that after this precharge takes place the information in memory cell 1 is to be read out. The output of the X decoder is therefore activated and a signal is applied to the gate of field effect transistors 9 and 10. Assume that the voltage on AX is V_{DD}, while the signal $\overline{AX}$ is of opposite polarity. Therefore word line 1 is activated while line 2 is not. A signal is therefore applied to the gate of the field effect transistor in the memory cell 1. The field effect transistor in this memory cell therefore becomes conducting. The data that are stored in the storage capacitor C1 is therefore transferred to the digit line or bit line 3. The potential on the other bit line 4 does not change. There is therefore a potential difference of the voltage on the bit lines 3 and 4. The function of the sense amplifier, as the name implies, is to amplify the potential difference and apply a potential to the digit lines. Since the Y decoder is activated at the same time as the X decoder the transfer field effect transistors then pass complementary outputs OUT and $\overline{OUT}$ to a main amplifier elsewhere on the circuit. These signals are then transferred to external circuitry.

The next consideration is how information is written into the memory cell. Writing takes place by bringing the address signal A_X to level V_{DD}. The field effect transistor in cell 1 is turned on and the transfer FETs are likewise turned on by the Y decoder. The input/output lines are then used for providing an input to the bit lines. A true value appears on the input line AN and a complementary value is then applied to the bit line 3.

Static MOS memory cell designs have generally not undergone as rapid a change as dynamic memory cell designs. One of the early static MOS memory cell designs utilized six MOS devices connected in a fairly elaborate circuit configuration that utilized a cross-coupled, flip-flop type of design. Another design utilized four MOS devices and two resistors (instead of two MOS devices) in its six device static memory cell. The purpose and function of the two resistors (or two MOS devices) are to act as load devices and to hold the charges or stored memory state in the cross-coupled MOS or FET devices. The other two MOS devices are used to perform read and write operations.

A later static MOS memory cell design was developed that reduced the size of the six-device static MOS memory cell to a four-MOS-device memory cell. However, this type of memory cell needed three different potential sources (0, -8, and -16 V) to operate, thus requiring two separate power supplies (-8 and -16 V).

Another perspective on memory technology is to consider the cell size of a random access memory cell as a function of time, with representations from different generations of semiconductor technology. Figure 8.7 shows the cell size of a dynamic RAM from the 1103 to the latest technology, represented by the Texas Instruments 64K 4164. It is seen that the cell size of the new double-layer silicon, scaled-down technology is almost one-tenth of the size of the 1103 memory cell.

In addition to the improvements in circuit design noted in connection with Figure 8.6, there have also been improvements in process technology resulting in shrinking cell sizes. Figure 8.8 shows the improvement in density made possible by increased levels of poly-

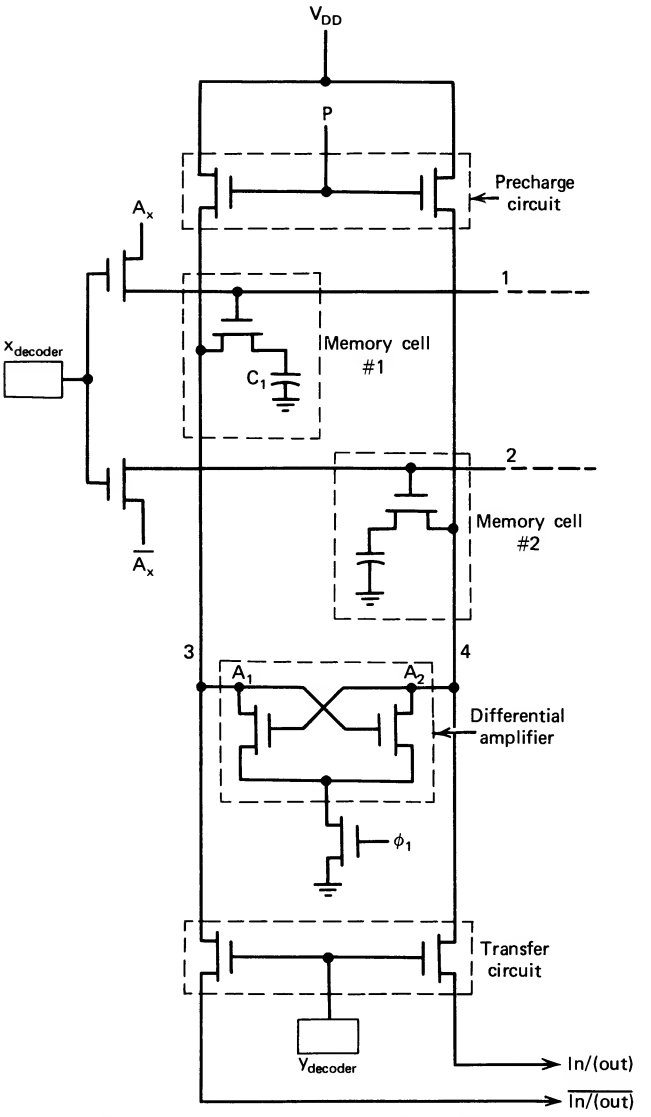


FIGURE 8.6 One-transistor dynamic memory cell.

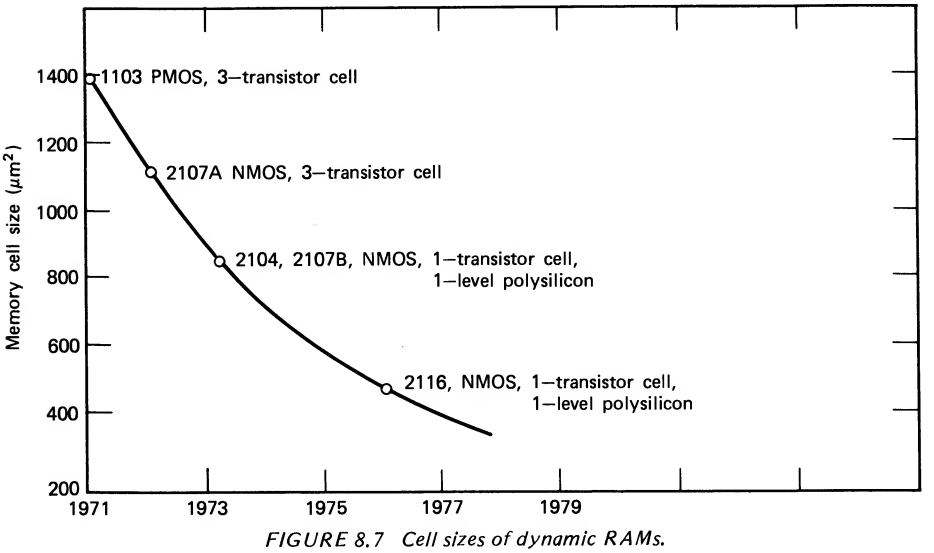


FIGURE 8.7 Cell sizes of dynamic RAMs.

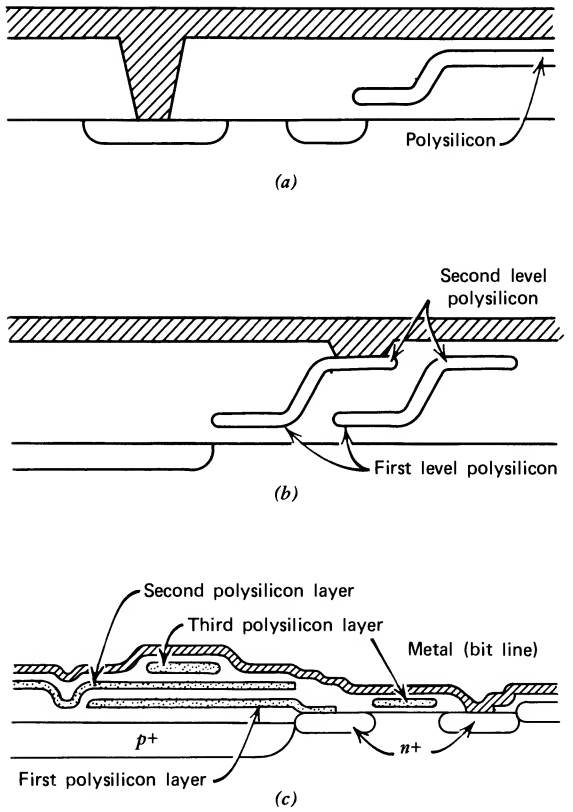


FIGURE 8.8 (a) *Single level polysilicon design.* (b) *Double level polysilicon design.* (c) *Triple level polysilicon design.*

silicon. Figure 8.8*a* shows an old single-level polysilicon design, Figure 8.8*b* a double-level polysilicon design, and Figure 8.8*c* a three-level polysilicon design. The new 16K RAMs are produced using double-level polysilicon, and the three-level polysilicon design is one technique presently under consideration for 64K and greater bit density designs. One three-level polysilicon design is used by Xicor, Inc. in the X2201 nonvolatile random access memory.

The problem of field inversion should also be mentioned. Field inversion occurs in MOS integrated circuits because of the presence of field oxide, thick oxide regions formed over the surface of the substrate surrounding each individual MOS transistor. If the voltage level on the electrical leads interconnecting the individual MOS transistors over the field oxide is sufficiently high, a conducting channel may be formed where none is intended, that is, a conducting channel between adjacent MOS transistors that are meant to be independently operated. The resultant parasitic transistor is called a field inversion transistor and the process itself is called field inversion.

The problem of field inversion is significant in *n*-channel MOS devices because the ratio between the threshold voltage caused by undesirable field inversion and the threshold voltage that is associated with the inversion under the gates of the transistor in an MOS device is considerably lower than for *p*-channel devices. There are various techniques to avoid the field inversion problem. One is to increase the thickness of the field oxide compared to the thickness of the gate oxide, thereby increasing the ratio of the field and gate threshold voltages. Another technique is to lower the operating gate voltage of the device. The disadvantage of the latter technique is that it decreases the operating performance. Yet a third technique is to adjust the doping concentration of the substrate underneath the field oxide, that is, to a level corresponding to less than 1 ohm/cm of resistivity. In actuality such a high adjustment of the bulk substrate doping level cannot be affected because of "body effect" and the resulting reduction in mobility, which lead to overall degraded performance.

Another approach is highly selective and accurate ion implantation, so as to have the bulk of the substrate at a relatively low doping concentration immediately below the field oxide. In such a technique it is necessary to mask the region where the implant is not wanted with photoresist. In such a case it becomes difficult to align the implant region with the mask used in subsequent photomasking steps. Yet another approach is to use silicon nitride masking. However, silicon nitride masking is less reliable and more difficult than processes using only silicon oxide mask.

Configuration. At the present time, the most popular configuration for semiconductor memory is the 4K RAM. The pin configurations for various types of 4K RAMs are shown in Figure 8.9, ranging from a 22-pin to a 16-pin package.

One of the first considerations in selecting a RAM is the package pin count. There is a trade-off involved with the package pin count. The smaller 16- and 18-pin packages require some type of multiplexing to store the same amount of data and accomplish the same functions as packages with a greater number of pins. On the plus side, the smaller package size means that a greater number of packages may be put in a given area on a printed circuit card. The actual difference in area is shown in Figure 8.10.

Another important point of comparison is the pin layout. Similar pin layout can mean plug compatibility for different memory chips, even though the chips may have different operating characteristics.

It would be useful at this point to illustrate some of the pin layouts for various types of RAMs. As a point for comparison, we present first an older device, the

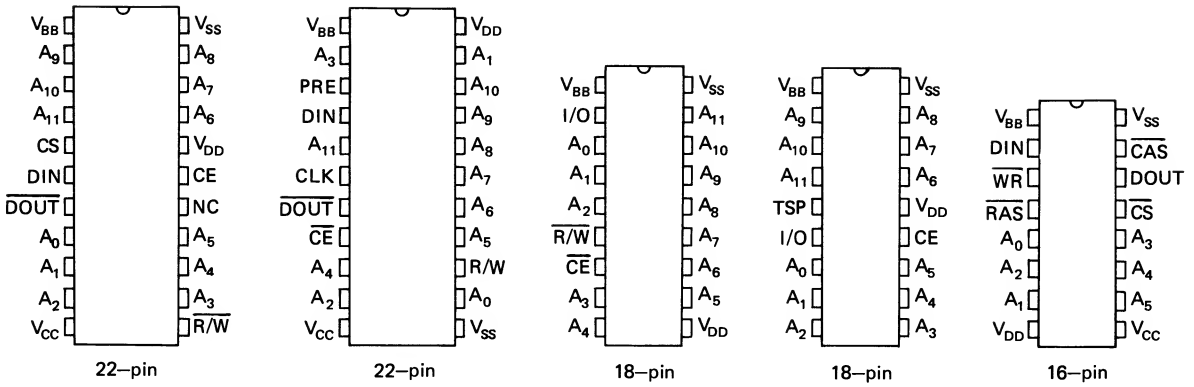


FIGURE 8.9 Configurations and sources for 4K dynamic RAM pin. (Courtesy Texas Instruments, Incorporated.)

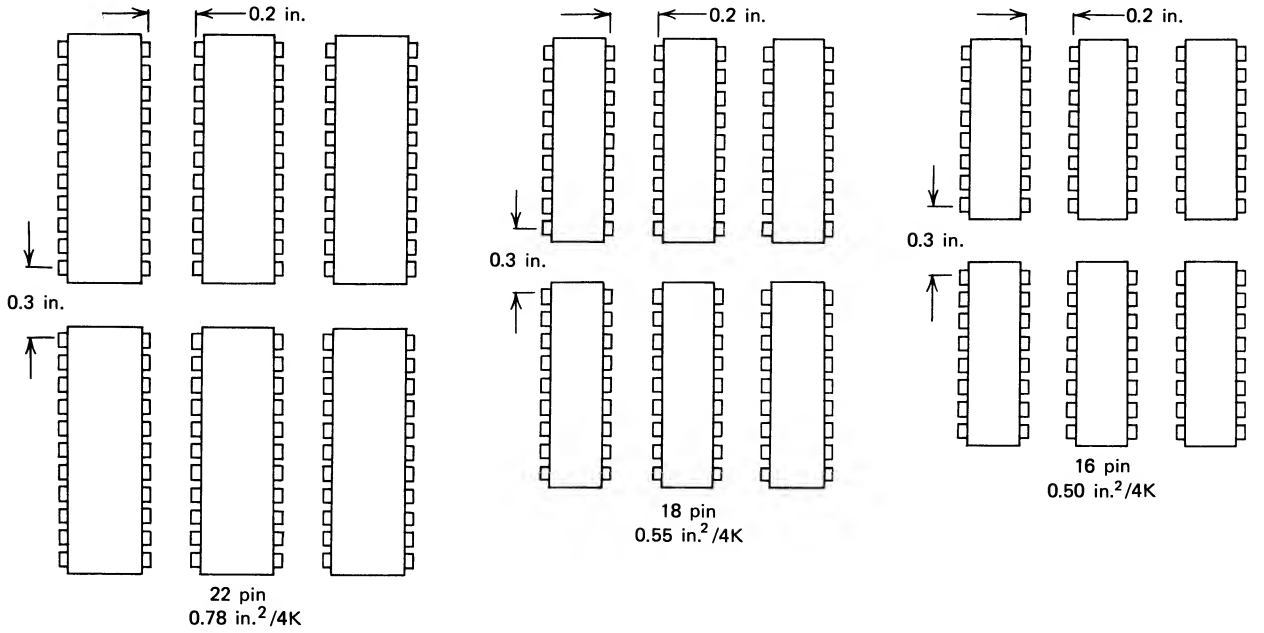


FIGURE 8.10 Board area for 22-, 18-, and 16-pin packages. (Courtesy Texas Insturments, Incorporated.)

Intel 2101, a 1024-bit static MOS RAM, packaged in a 16-pin package, in Figure 8.11.

The Mostek 4027, a very popular 4096-bit dynamic MOS RAM, also packaged in a 16-pin package, is shown in Figure 8.12.

Another important point concerning the pin configuration of RAM packages is the possible compatibility of both 4K and 16K devices. Figure 8.13 shows both a 4K and a 16K RAM package that are compatible: the chip select pin (CS) on the 4K package has been replaced by the A_6 address pin on the 16K package. Also shown is the proposed standard 64K RAM package. Thus with a properly designed circuit board, the user may upgrade the memory capacity of a system employing such 4K RAM chips by merely unplugging them and replacing them with 16K chips.

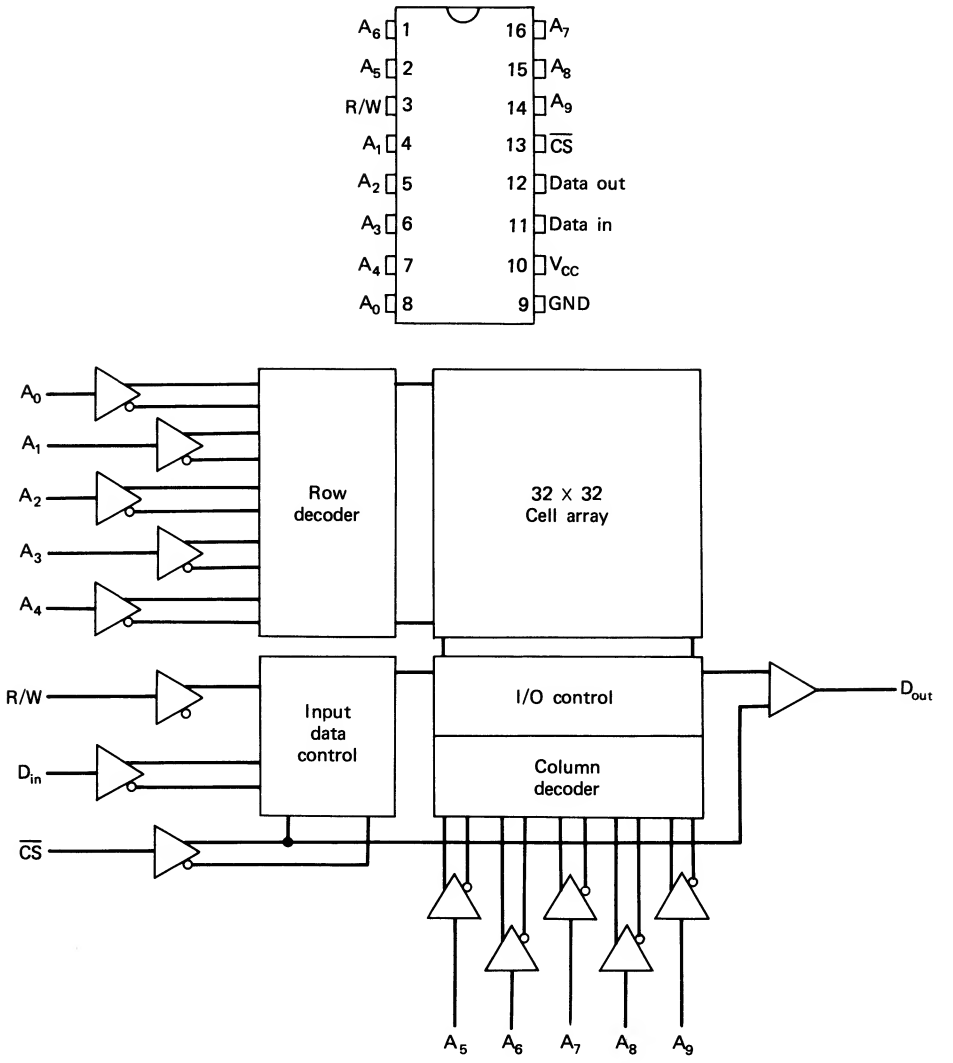


FIGURE 8.11 Intel 2101 1K static MOS RAM. (Courtesy Intel, Inc.)

The various types of commercially available RAMs are summarized in Table 8.1.

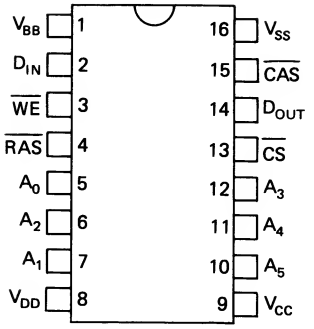
Interfacing of RAMs. Interfacing of RAMs generally refers to the interfacing of the microprocessor to the RAMs.

There are three types of interfacing methods possible depending on the type and degree of buffering, multiplexing, and device selection:

1. No buffering; multiplexed bus; linear device selection.
2. Single level of buffering (single card application); demultiplexed bus; decoded chip enables.
3. Demultiplexed bus; multiple levels of buffering (as in a multicard system); decoded chip enables.

The selection of one of these interfacing systems depends on the particular system configuration required by the user. The critical parameter that must be considered from the point of view of interface design is the access time for both the memory and the processor. For the processor and the memory to interface correctly, the access time of the memory must be compatible with the access time of the processor. The access time of the processor is the time interval measured from the presentation of a control signal to the data bus to the time that the processor expects the data on the data bus to be valid. The access time for the memory is the interval between the time when the memory is requested to present the data and the time when the data are actually presented.

A typical memory system using 16-pin 4K bit memory



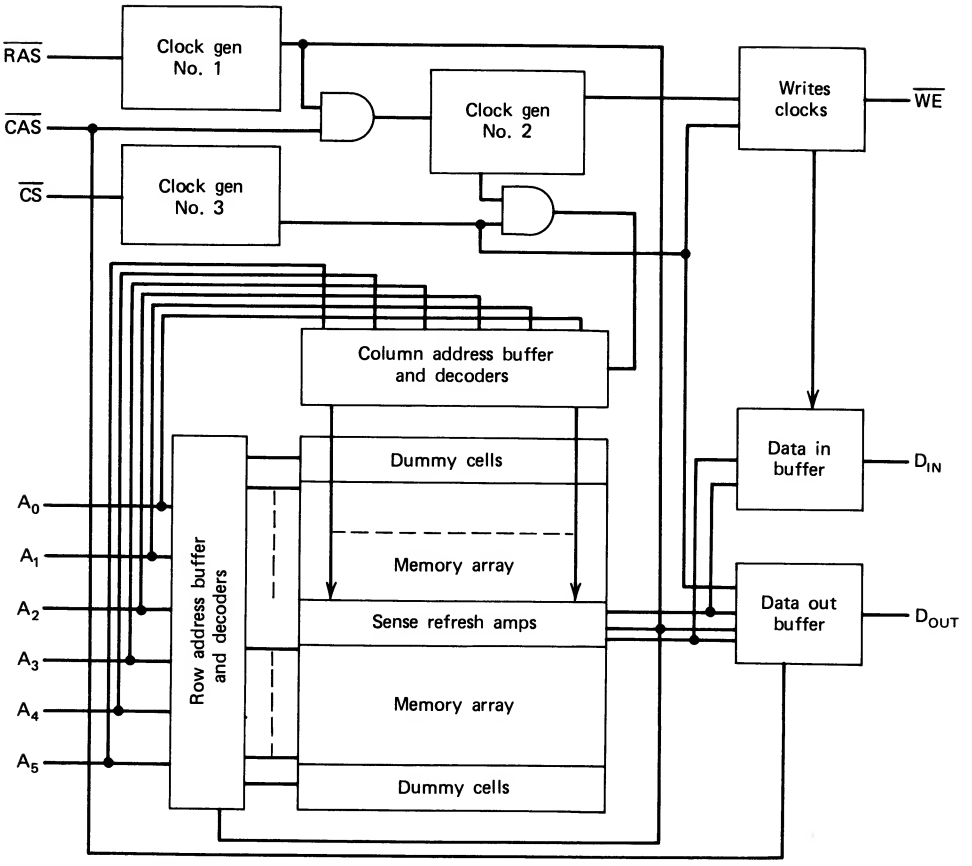


FIGURE 8.12 Mostek 4027 4K dynamic MOS RAM. (Courtesy Mostek.)

Table 8.1. Typical NMOS RAMs

Type	*Part Number*	*Organization*	*Access Time (nsec)*	*Pins*
Dynamic	MK 4027	4096 X 1	150	16
	MK 4116	16384 X 1	120-200	16
	TMS 4164	65536 X 1	150	16
Static	Intel 2114	1024 X 1	200-450	18
	Intel 2118	2048 X 8	100-190	16
	Intel 2167	16384 X 1		20

devices is shown in Figure 8.14. This system is a 16K word system, the address lines being A0-A11, the data input lines DI1-DI8, and the data output lines DO1-DO8.

Mass Memories

Interfacing with mass memories is equally as important as interfacing with RAM because most systems incorporate some type of mass storage. In the present section we consider the following types of mass storage:

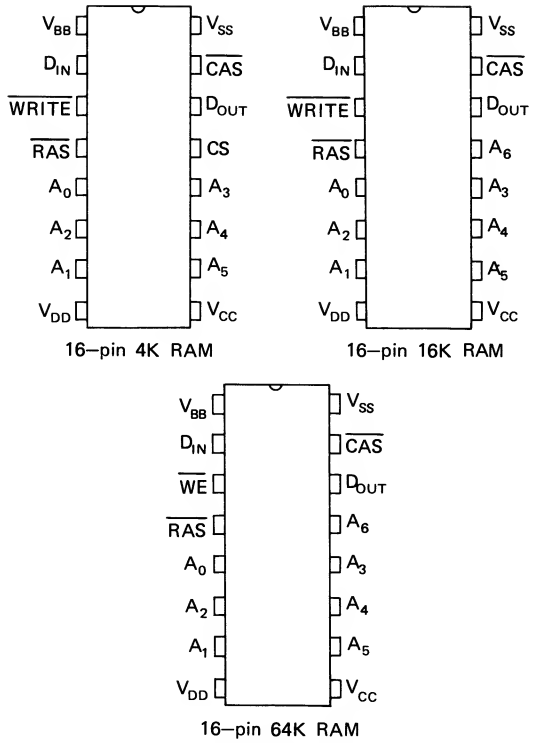


FIGURE 8.13 RAM packages of 4K, 16K, and 64K.

- Floppy disks.
- Fixed disks.
- Magnetic bubble memories.
- Charge transfer devices.

Floppy Disks. The floppy disk is the most popular external storage device for a microcomputer system. Before considering the interfacing issue, it would be worthwhile to summarize some of the different types of floppy disks and their features.

First, there are two basic sizes of floppy disks: the standard floppy and the minifloppy. The distinctions between the standard floppy and the minifloppy are highlighted in Table 8.2. The remainder of this section is reprinted with permission from McGlynn: *Personal Computing.* (Wiley, New York, 1979).

"The standard floppy disk was originally developed by IBM as part of the IBM 3740 key-to-disk data-entry system. The 3740 recording format, shown in Figure 8.15, in which the disk is divided into 77 tracks, with each track consisting of 26 sectors, became the standard recording format. Each sector has 128 bytes in one format and 256 bytes in another format. Various non-IBM formats may also be defined having a sector length ranging from 16 to 4096 bytes.

"There are two basic techniques of "sectoring" the

Table 8.2. Minifloppy and Standard Floppy Disk Characteristics

Characteristic	*Minifloppy*	*Floppy Disk*
Unformatted capacity (bytes)	110K	400K
Formatted capacity (soft sectoring) (bytes)	80.6K	253K
Tracks	35	77
Sectors/track	18	26
Capacity/sector (bytes)	128	128
Transfer rate (bps)	125K	250K
Average access time (sec)	0.566	0.286
Rotational speed (rpm)	300	360
Head loading time (msec)	75	35
Recording density (max bpi) (single density)	2600	3200
Typical drive size (inches)	3.25 X 5.75 X 8.0	4.62 X 8.55 X 14.25
Typical weight (lb)	3	14
Typical dc power required (V)	5, 12	-15, 5, 24
Typical drive price (quantity 1)	$390.	$600.
Medium-size (inches)	5.25 X 5.25	8.0 X 8.0
Medium price (quantity 1)	$4.50	$6.50

disk: hard and soft. Hard sectoring uses physical holes in the disk itself to mark the beginning of each sector. Soft sectoring uses just one hole in the disk to mark the beginning of sector 0; the remaining sectors are defined by the user in software by appending a header to each sector that identifies it.

"Most of the disk systems used with microcomputers are soft sectored. Software is readily interchangeable among users with soft sectored disk systems. There are a few hard sectored disk systems for microcomputers—the Processor Technology Helios system, the North Star mini-floppy, and the Micropolis system. The advantage of hard sectoring is that the disk is able to store more user data than the soft sectored disks, which include a considerable amount of overhead. The "overhead" on a soft sectored disk is shown more explicitly in Figure 8.16.

"Figure 8.16 shows the track format for the soft sectored, 128 byte per sector IBM format. Note that the basic fields consist of a gap, followed by an identification field, followed by another gap, followed by data [including a cyclic redundancy check (CRC) error detection field], followed by a gap.

"There are at least three different recording techniques used with floppy disks, as shown in Figure 8.17. The most common is frequency modulation (FM), shown in Figure 8.17*a*. A variation of FM, known as modified frequency modulation (MFM), is shown in Figure 8.17*b*, and yet another variation called modified-modified

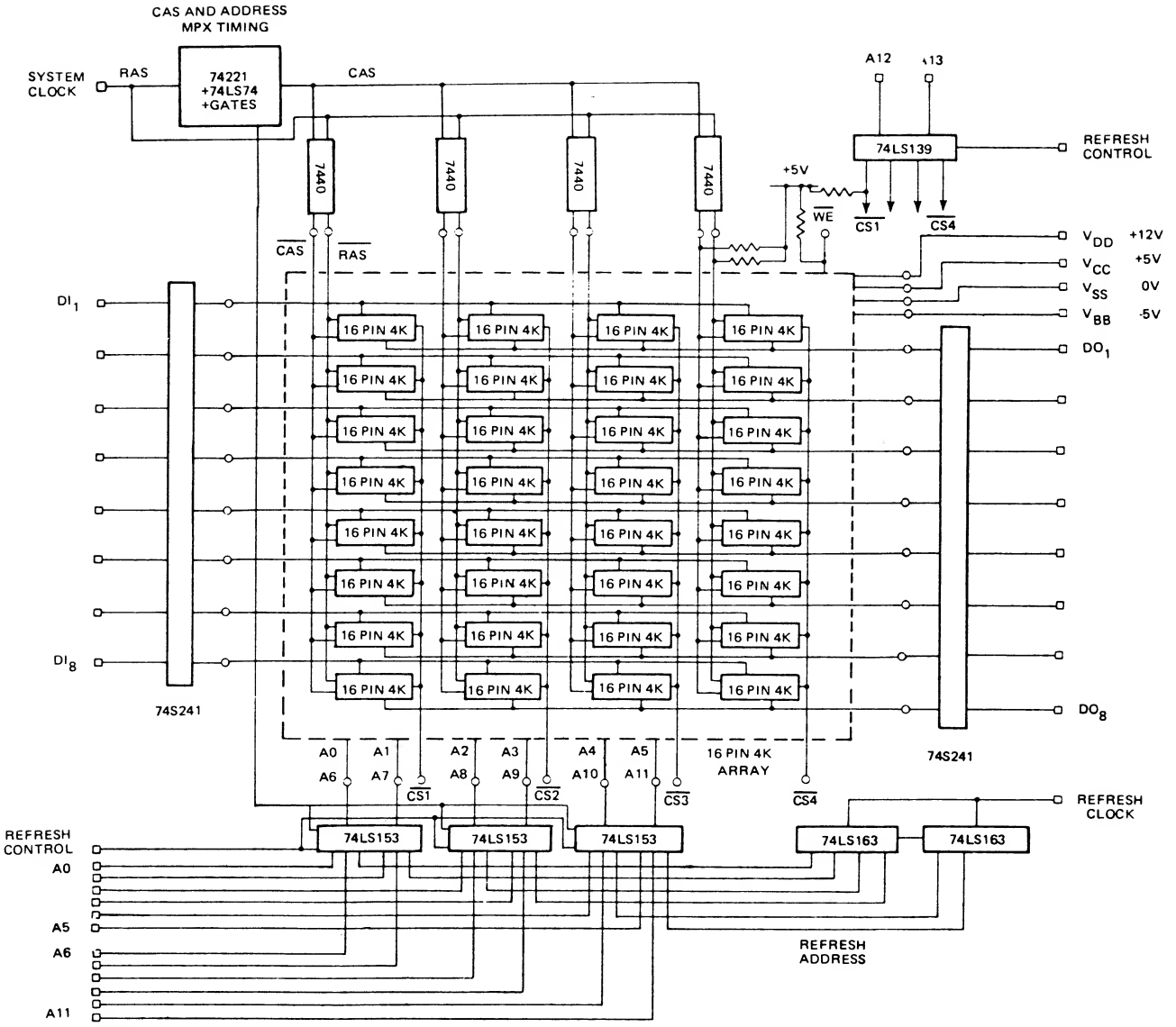


FIGURE 8.14 A 16K word memory system using 4K bit dynamic RAMs. (Courtesy Texas Instruments, Incorporated.)

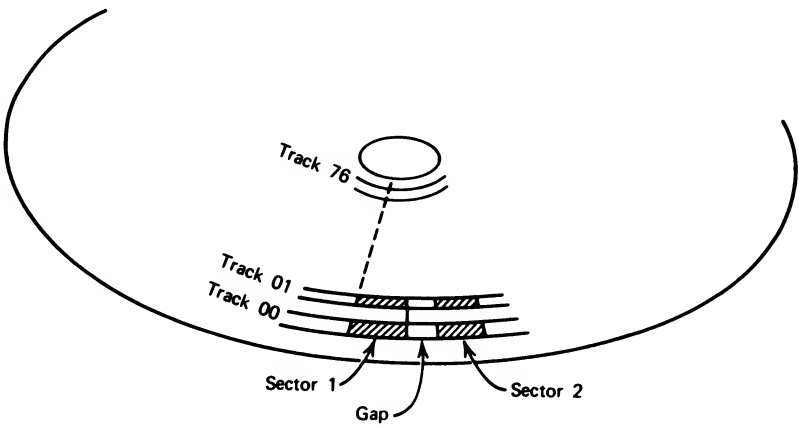


FIGURE 8.15 IBM 3740 floppy disk recording format.

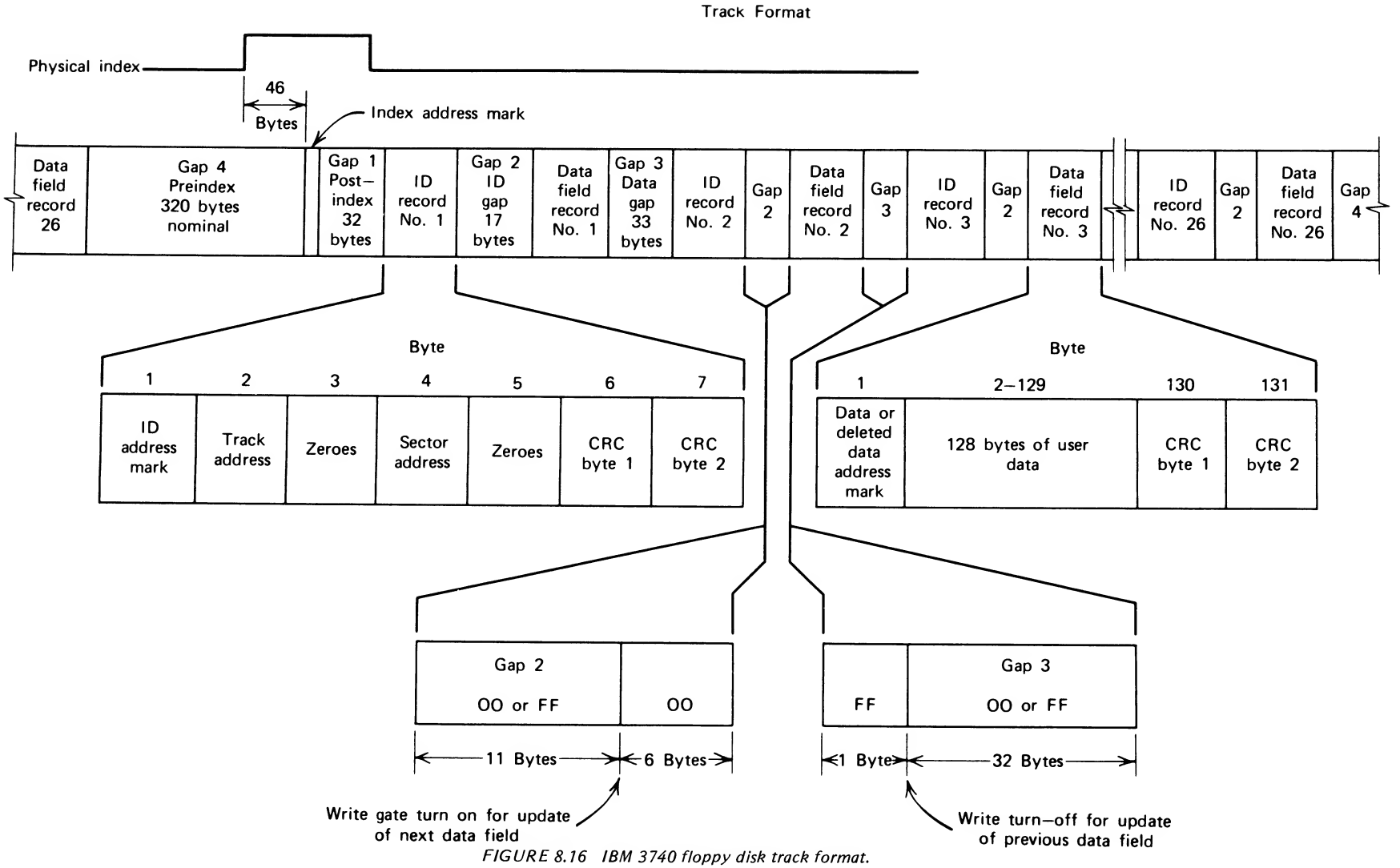


FIGURE 8.16 IBM 3740 floppy disk track format.

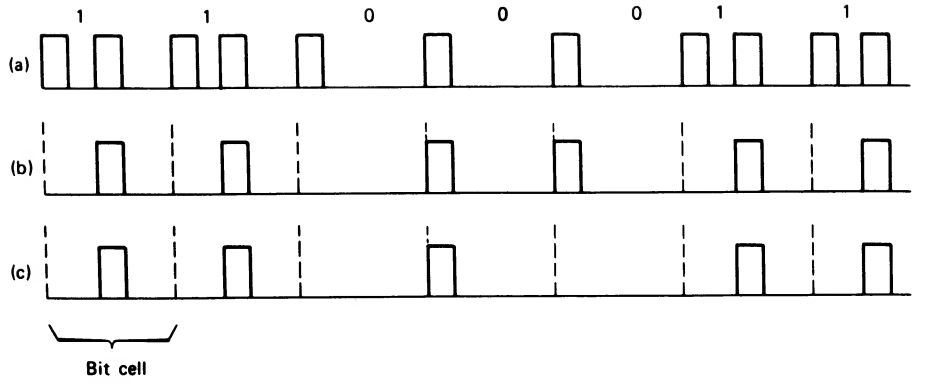


FIGURE 8.17 Floppy disk recording codes: (a) *frequency or double frequency modulation (FM);* (b) *modified frequency modulation (MFM);* (c) *modified modified frequency modulation (M^2FM).*

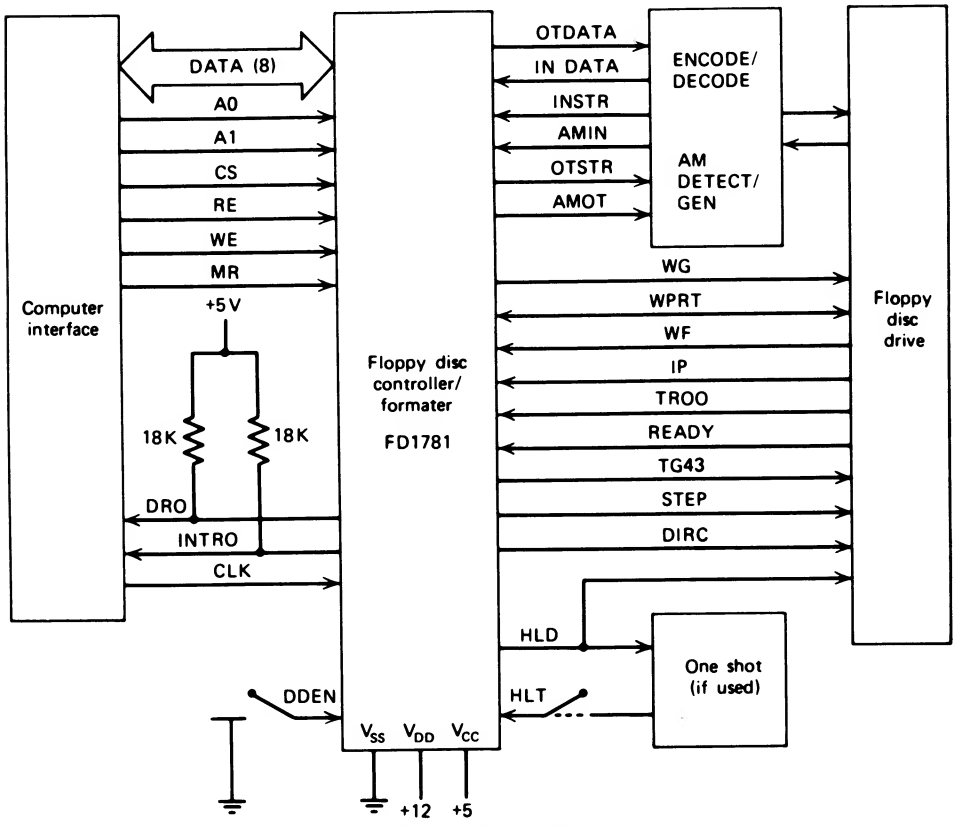


FIGURE 8.18 Floppy disk interface.

frequency modulation (M^2FM) is shown in Figure 8.17*c*.

"The basic block diagram of a floppy disk interface is shown in Figure 8.18. The floppy disk drive is capable of performing the following operations: move the head to a specific track, read or write data from the track, and monitor control signals.

"The floppy disk controller or formater is designed to interface the computer with the floppy disk drive. Such controllers are typically implemented on a single printed circuit card. Recently, a number of single chip controllers have been introduced, such as the Western Digital FD 1781 shown in Figure 8.19. The block diagram of the FD 1781 is also shown in Figure 8.18.

"Using Figure 8.18, we can briefly explain the function of the signal lines between the floppy disk controller and the computer interface. Beginning at the top, the eight-data line form a bidirectional bus for transferring data, control, or status signals between the computer and the

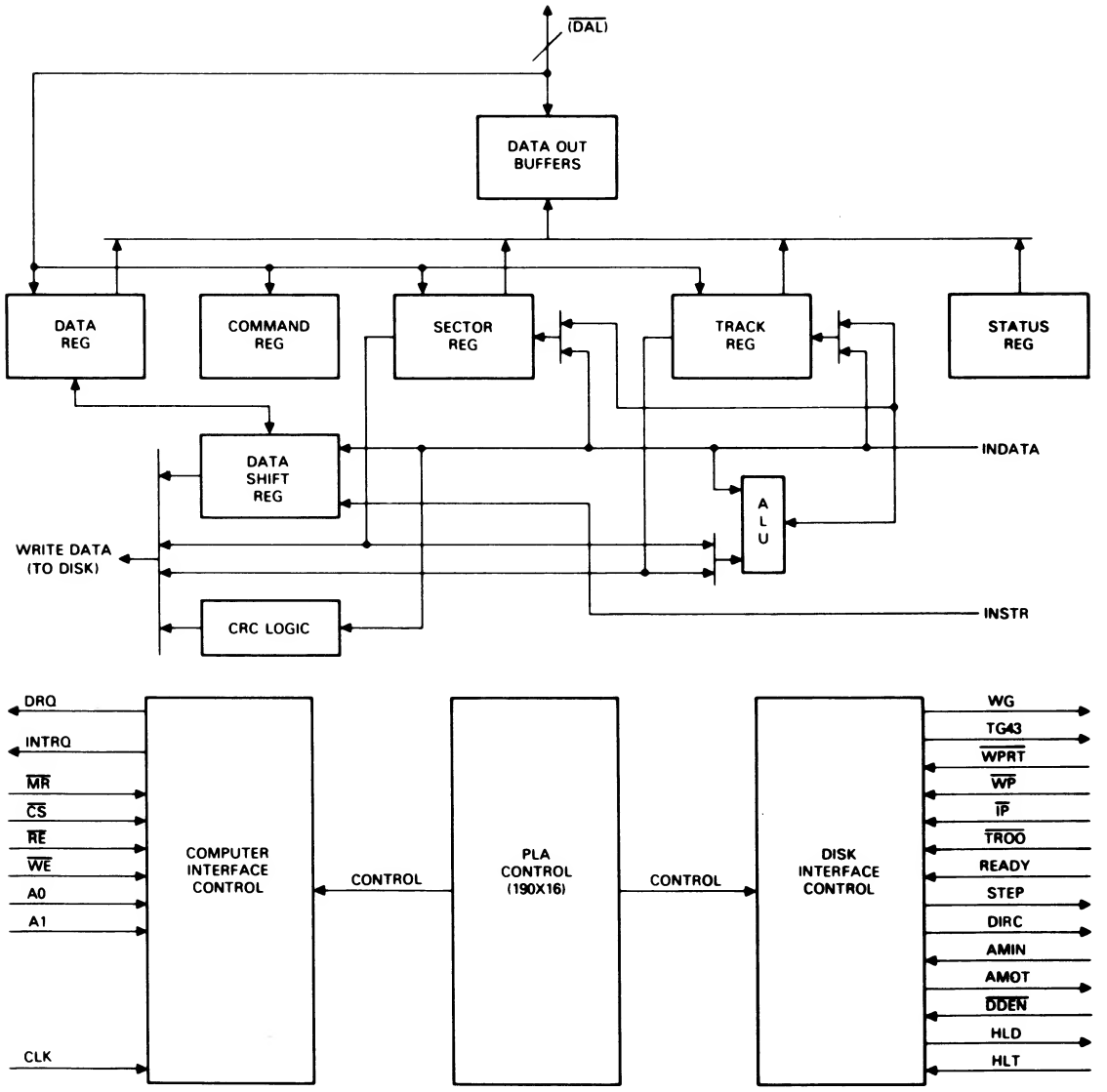


FIGURE 8.19 Western Digital FD 1781.

controller. The A0 and A1 lines are register select lines that specify which register in the controller is to receive or transfer data. As shown in the block diagram in Figure 8.19, these registers are a data register, a command register, a sector register, a track register, and a status register. The CS line is the chip select line that enables computer communication with the controller. The RE line is the read-enable and the WE line is the write-enable line. MR is a master reset, DRQ is a data request, INTRQ is an interrupt request, and CLK is the clock line.

"On the floppy disk side of the controller, OTDATA is the output data to the floppy disk drive from the controller, and INDATA is input data from the floppy disk drive. INSTR is the input strobe line to indicate that the signal on the INDATA is valid. AMIN is an address mark detector, OTSTR is the output strobe line, and AMOT is the address mark output line. WG is the write gate line, which is valid when writing is to be done on the floppy disk. WPRT is a write-protect line, WF a write-fault line, IP an index-pulse line, TR00 the track 00 signal line, and READY indicates the disk's readiness to read or write. TG43 indicates that the read/write head is positioned between track 44 and 76. STEP and DIRC are the step and direction motor controls for controlling the stepping motor of the disk drive. HLD is a head load signal line, and HLT is a head load timing signal. DDEN is the pin that selects whether single density or double density recording is to be performed."

Fixed Disks. The fixed or nonremovable disk is be-

Table 8.3. Eight-Inch Fixed Disk Characteristics

Manufacturer	*Model No.*	*Capacity (m bytes)*	*Average Access Time (msec)*	*Transfer Rate (bytes/sec)*	*Bit Density (bpi)*	*Track Spacing (tpi)*	*Media Size (mm)*	*Introduced*
BASF	6171, 6172	8, 24	50	800K	6542	500	210	mid-1979
IMI	7710, 7720	11, 20	50	648K	6000	300	200	1980
Kennedy	7000	4, 12	50	550K	6000	300	210	late 1979
Memorex	101	11.7	70	593K	6100	195	200	1980
Micropolis	1201, 1202, 1203	9, 27, 45	34	743K	6100	195	200	1980
New World	211	2	18	605K	8000	100	200	1980
Pertec	D8000	20	50	870K	6688	476	210	1980

coming increasingly important in microcomputer systems requiring mass memory in the megabyte range. Although large fixed disk drives have been around for mainframe systems for some time, the new 8-in. fixed disk, first developed by IBM under the code name Piccolo and introduced as the 3310 on the System/34 in February 1979, is attracting the most interest for small computer systems. Representing later technology, the 3310 has greater track and bit densities than the earlier and now standard 14-in. drives. Table 8.3 presents some of the announced 8-in. fixed disk drives and some of their characteristics. In single quantities the announced prices of the drives range from $2000 to $3000.

The Pertec D8000, shown in Figure 8.20, is a typical example of the 8-in. fixed disk drive. The D800 uses two aluminum alloy disks with three data surfaces and one prewritten servo surface. The oxide disk surface is coated with a lubricant that allows the read/write head to actually rest on the media. The disk itself is sealed in an environmental container in a "clean room" environment during manufacture, thereby eliminating the danger of contaminants. Such a sealed disk is similar to the original IBM 3348 "Winchester" disk, and thus the "Winchester" appelation has been applied to many of the 8-in. disks by various vendors.

FIGURE 8.20 Pertec D8000 8-in. fixed disk. (Courtesy Pertec Computer Corporation.)

Magnetic Bubble Memories. Magnetic bubble technology is an important data storage technology that should be considered here. Magnetic bubbles are small, cylindrical "domains" of magnetization whose magnetic polarization is opposite to that of the thin magnetic film in which they are imbedded. The bubbles are formed and are stable when a magnetic bias field in a predetermined range is applied perpendicular to the layer, such as shown in Figure 8.21.

Magnetic bubbles can be used to represent data because they can be moved or "propogated" from one point to another along the layer of magnetic material. The presence or absence of a bubble at a predetermined position in a sequence of bubbles is used to represent a binary 0 or 1.

The bubble diameter is approximately equal to the thickness of the layer of magnetic material, so in order to achieve a high packing density extremely thin magnetic films (from 1 to 4 microns in thickness) are utilized. Typical magnetic films are formed from rare earth orthoferrites, rare earth aluminum, and gallium substi-

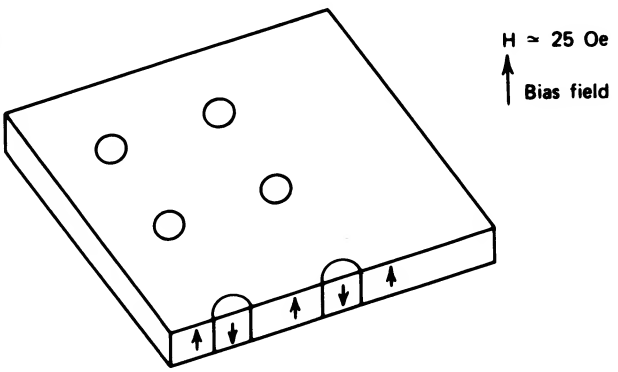


FIGURE 8.21 Magnetic bubble domains.

tuted iron garnets. The magnetic layer is epitaxially grown on a nonmagnetic garnet substrate. The typical substrate used is gadolinium gallium garnet.

There are two basic techniques by which magnetic bubbles can be moved or propagated in the layer of magnetic material:

- Field accessing.
- Current (or conductor) accessing.

Field accessing utilizes a guide structure typically configured as a repetitive pattern on the surface of the magnetic layer. The guide structure is usually formed by a pattern of elements of magnetically "soft" material, such as Permalloy, but may be formed by other structures such as ion implanted contiguous disks as well. A rotating in-plane magnetic field is then applied parallel to the surface of the layer so that the guide structure elements assume local specific magnetic polarities in response to the changing orientation of the external magnetic field. The rotating field is typically produced by coils of wire surrounding the magnetic bubble domain chip. A series a travelling potential wells is therefore set up in the magnetic layer that causes the bubbles to propagate. The typical configuration of the Permalloy guide structure includes repetitive T-I bar, Y-bar, or chevron patterns.

Current accessing utilizes two or more conductor patterns on the surface of the magnetic layer that are driven by a polyphase current. Current access devices are more difficult to fabricate because they require two or more levels (and process steps) over the entire device area.

Figure 8.22 illustrates how the T-I bar pattern permits the propagation of magnetic bubbles as the external magnetic field rotates in the plane of the layer. The magnetic bubble has a given magnetic polarity on the top surface of the layer (e.g., "north" polarity). The magnetic polarity of adjacent portions of the guide elements change with the external magnetic field, and the bubble is attracted to the closest guide element portion with a "south" magnetic polarity. As the in-plane magnetic field continues to rotate, the bubble is propagated along the guide structure as shown in the figure.

The field access guide structures are used on all commercial bubble domain devices today. The guide structures are organized into regions for bubble generation, replication, and storage. The typical architecture employed is in the form of a major loop, minor loop storage hierarchy. A major loop/minor loop configuration is an organization in which a pattern of Permalloy elements defines a plurality of closed loop paths called minor loops about which the bubbles move, and an accessing path called the major loop. The minor loops are

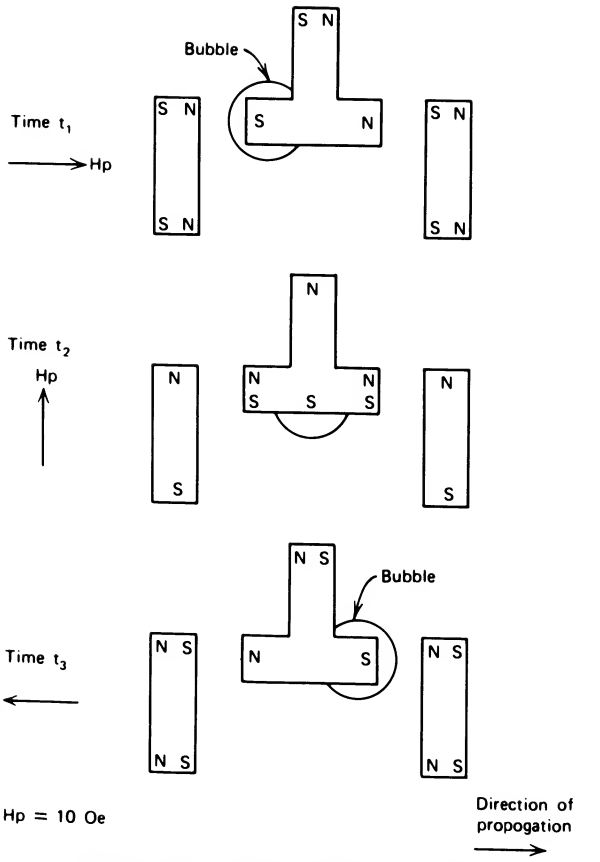


FIGURE 8.22 Magnetic bubble propagation.

operative to store data while the accessing path or major loop intersects one of the elements in each minor loop so that bubbles may transfer from the minor loop to the major loop at appropriate times. Such transfer is achieved by means of two-way transfer gates that permit the transfer of magnetic domains from the minor loop to the major loop and the major loop to the minor loop. Data stored in the minor loops is circulated until the desired word (composed of one bit from each of the minor loops) reaches the transfer gate. On command of a transfer signal, the information is transferred to a major loop whereupon it may simply be read out or erased or erased and overwritten. The data may also be further advanced along the major loop in response to the rotating magnetic field and transferred back into the minor loops.

The alternative form of magnetic domain propagation utilizes electrical conductors and is called current or conductor accessing. The conductors may typically be formed in multiple layers, insulated from one another and driven in a three-phase manner.

The first commercial bubble memory chips were implemented in Texas Instruments Silent 765 portable terminal in 1978. The terminal used a 92-kilobyte chip using a simple major/minor loop architecture and having

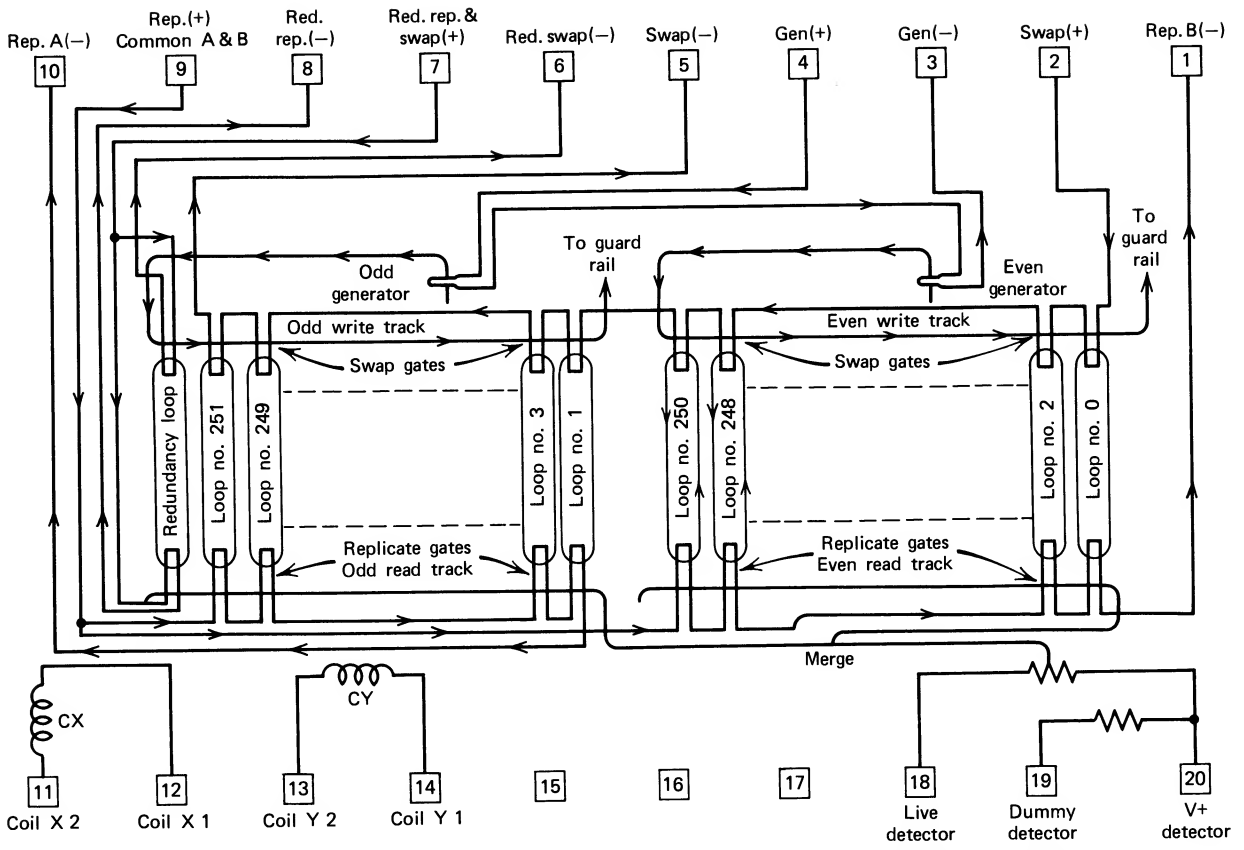


FIGURE 8.23 Texas Instruments TIB0303 magnetic bubble memory. (Courtesy Texas Instruments, Incorporated.)

a cycle time of 12.8 msec. The company later introduced a 256-kilobyte bubble memory chip having a "block replicate" Major/minor loop architecture. In the block replicate architecture, a bubble replicator is included in each minor loop so that instead of being transferred completely out of the minor loops, the bubbles are merely duplicated or replicated onto the output track during a read operation.

The block replicate architecture of the Texas Instruments TIB0303 is shown in Figure 8.23.

The interface for the Texas Instruments bubble memory system is shown in Figure 8.24. The controller implements a parallel to serial and serial to parallel version between the memory and external circuits. The controller furthermore contains a buffer and provides addressing and control outputs to the timing generator. The controller functions to start and stop the propagation of bubbles, the maintenance of the page position, and the setting of flags for various bubble memory functions.

Rockwell RBM256. Another bubble memory device is the Rockwell RBM256 shown in Figure 8.25. The device is packaged in an 18-pin DIP package about 1.2 X 1.2 in. The actual chip size is also shown in Figure 8.25, against a background of a highly magnified portion of the chip's circuitry, showing a chevron pattern. The RBM256 stores 1025 blocks of 260 bits and has an average access time of less than 4 msec. The internal architecture of the RBM256 is shown in Figure 8.26.

The RBM256 has a major/minor loop block replicate architecture. Data blocks of 260 bits are stored as bit positions on a plurality of loops, with the same relative bit position in each loop making up 1 bit of the block. Thus the block length is equal to the number of loops.

As shown in Figure 8.26, odd bits are stored on one side of the bubble memory chip, and even bits on the other. The output port is adjacent to an edge of each of the loops containing a replicate/transfer switch as well as magnetoresistive detectors. Each of the detectors consists of a stack of chevrons that stretches a bubble into an elliptical cylindrical domain the same width as a bubble, but having the length of the 260 chevron stack. According to the magnetoresistive effect, a change in resistance is induced in the detector, which is then translated into a difference signal between the current flow in an active detector and one in a reference or dummy detector containing no bubbles.

Since interfacing directly with a magnetic bubble de-

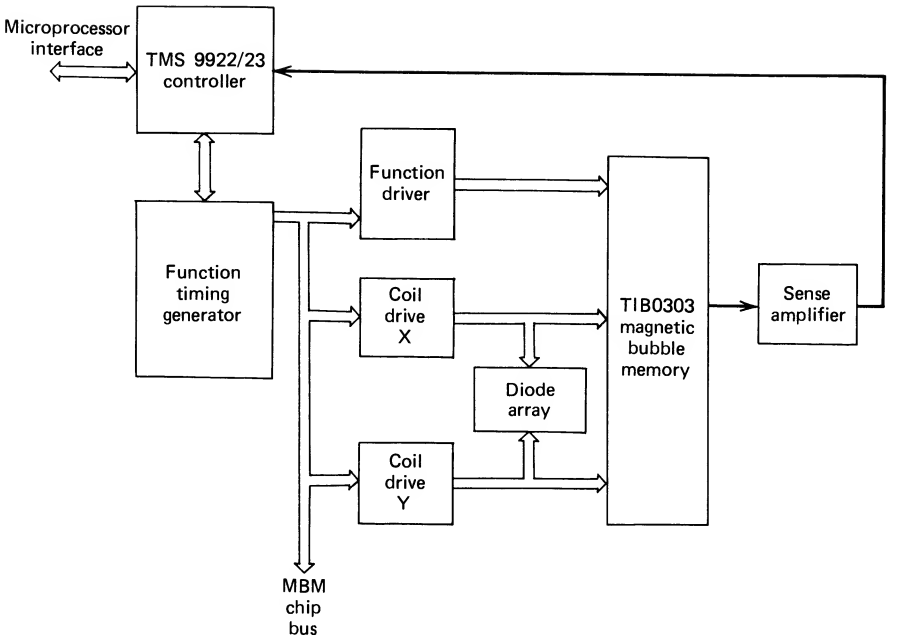


FIGURE 8.24 Texas Instruments magnetic bubble memory system. (Courtesy Texas Instruments, Incorporated.)

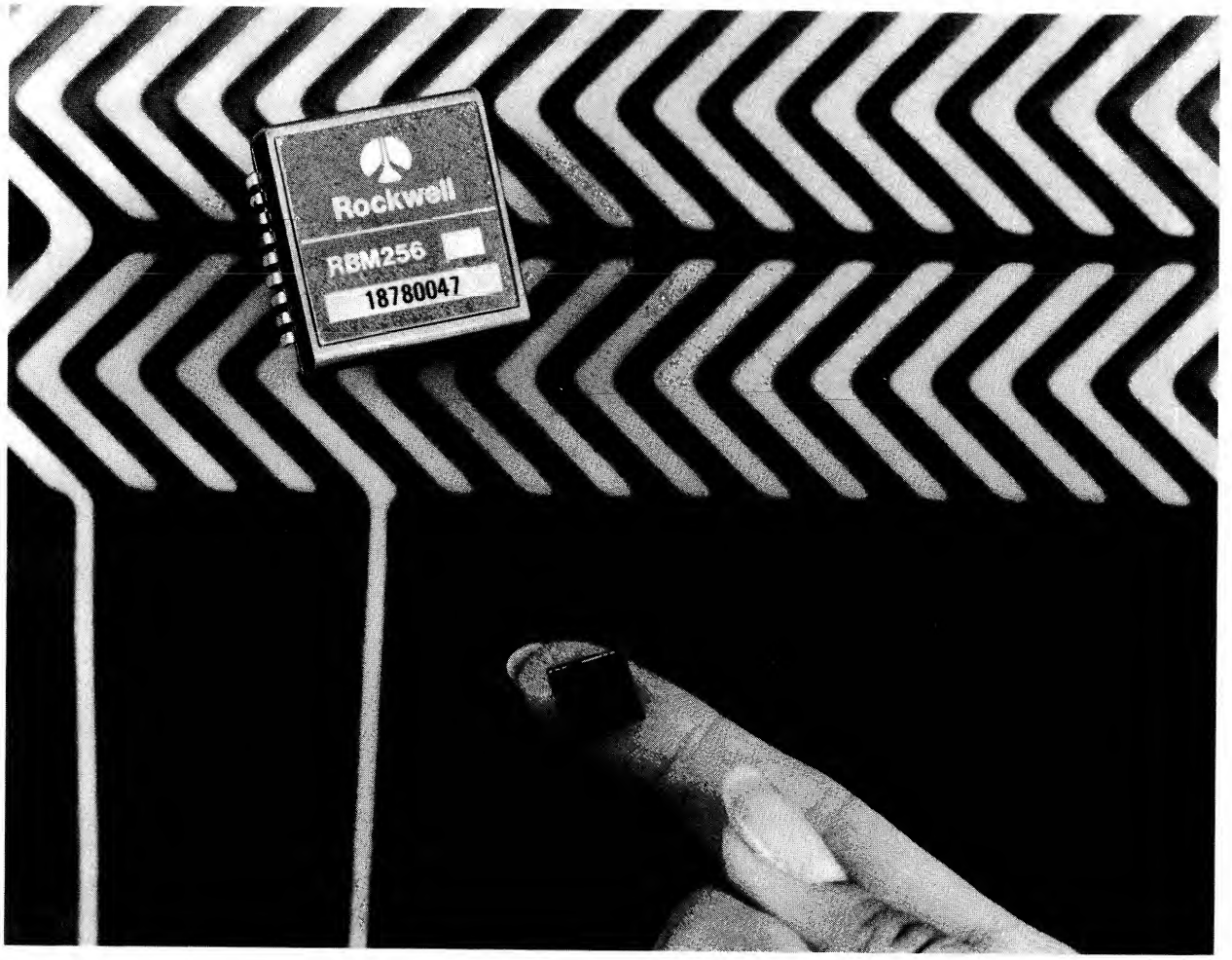


FIGURE 8.25 Rockwell RBM 256K bubble memory device. (Courtesy Rockwell International.)

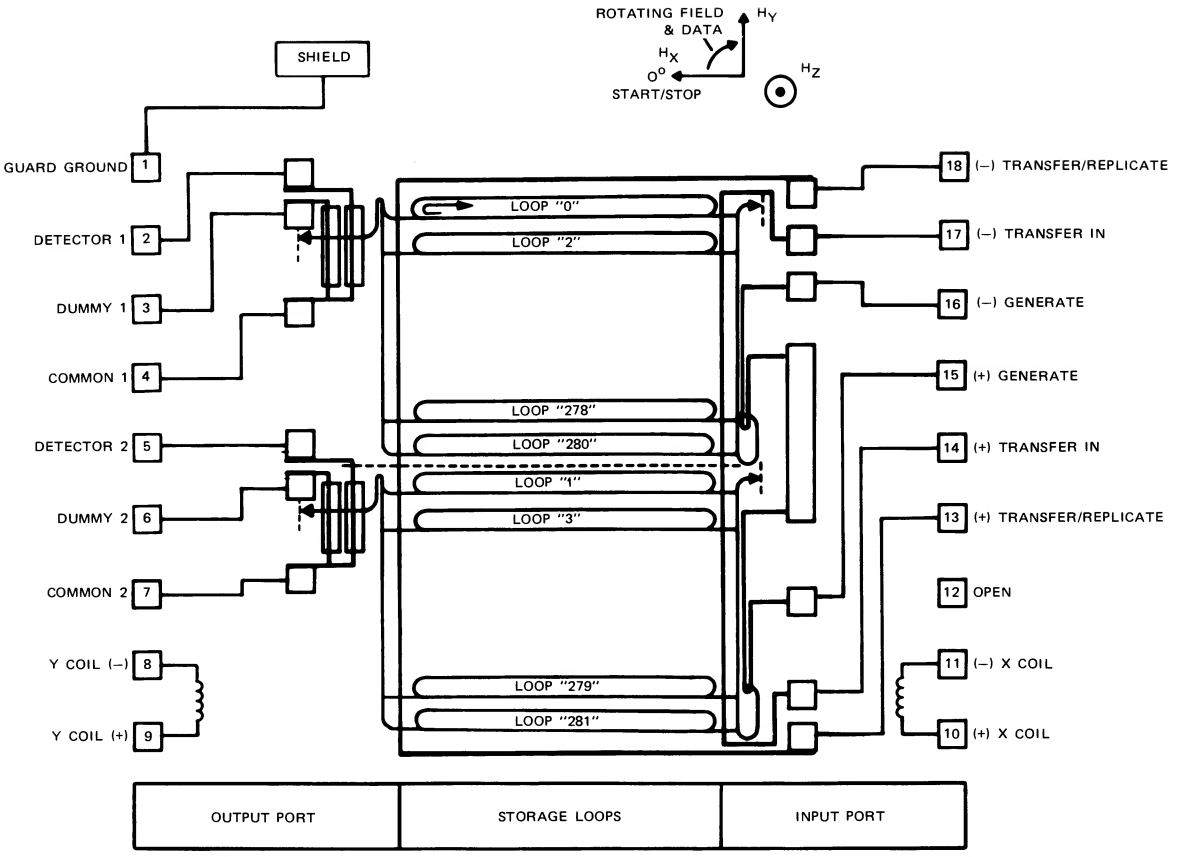


FIGURE 8.26 Rockwell RBM256 device architecture. (Courtesy Rockwell International.)

vice is a task that must be performed by all users, bubble memory devices are also available in card or module form to which the interfacing problem has been simplified. An example of such a module is the Rockwell 1-megabit bubble memory module shown in Figure 8.27 and in the block diagram of Figure 8.28. The 1-megabit memory module, known as part RLM658, includes four of the RBM256 bubble memory devices operating in parallel. The RLM658 is implemented on a PC card that is compatible with the Rockwell 65 development system based on the 6502 microprocessor. Also available is a control module that is software compatible with System 65 and that can control from 1 to 16 RLM658 modules.

It would be worthwhile to briefly review some of the timing and interface signals associated with the RLM658 module.

Figure 8.29 is a timing diagram of the coil drive signals applied to the bubble memory module. The first timing signal (labeled COILNF) is the coil enable signal that functions to activate the coil drive circuitry and initiate data movement. The COILNF signal typically remains on for several hundred cycles to permit access of an entire block of data. The four signals in Figure 8.29*a* labeled ODF, OAF, OBF, and OCF are the coil driver timing signals, each of which defines one quarter of an operation cycle. The two signals underneath the coil driver timing signals are the actual current wave forms that are generated by the coil driver unit in the module and that produce the signals that are applied to the two coils in the memory. It is noted that the two current wave forms are 90° out of phase.

The minimum run duration of the current wave forms are 509 cycles for writing and 366 cycles for reading one block of data. The speed of the bubble memory in terms of the access time is related to the frequency of the field coil. The unit under description operates at a frequency of 150 KHz so that the access time for reading averages 5.96 msec and that for writing averages 7.22 msec.

Figure 8.29*b* shows the read and write sequences for the module. The first signal, labeled TOF, is the transfer out signal, which when energized transfers a block of data from the storage loops onto the read track. The next timing signal shown is the R/WF or read/write signal, which activates the appropriate operator for performing a read or write function. To enter data into the memory, the address block location must be positioned at the transfer out location. Activation of the TOF signal empties the block of data on the read track and clears

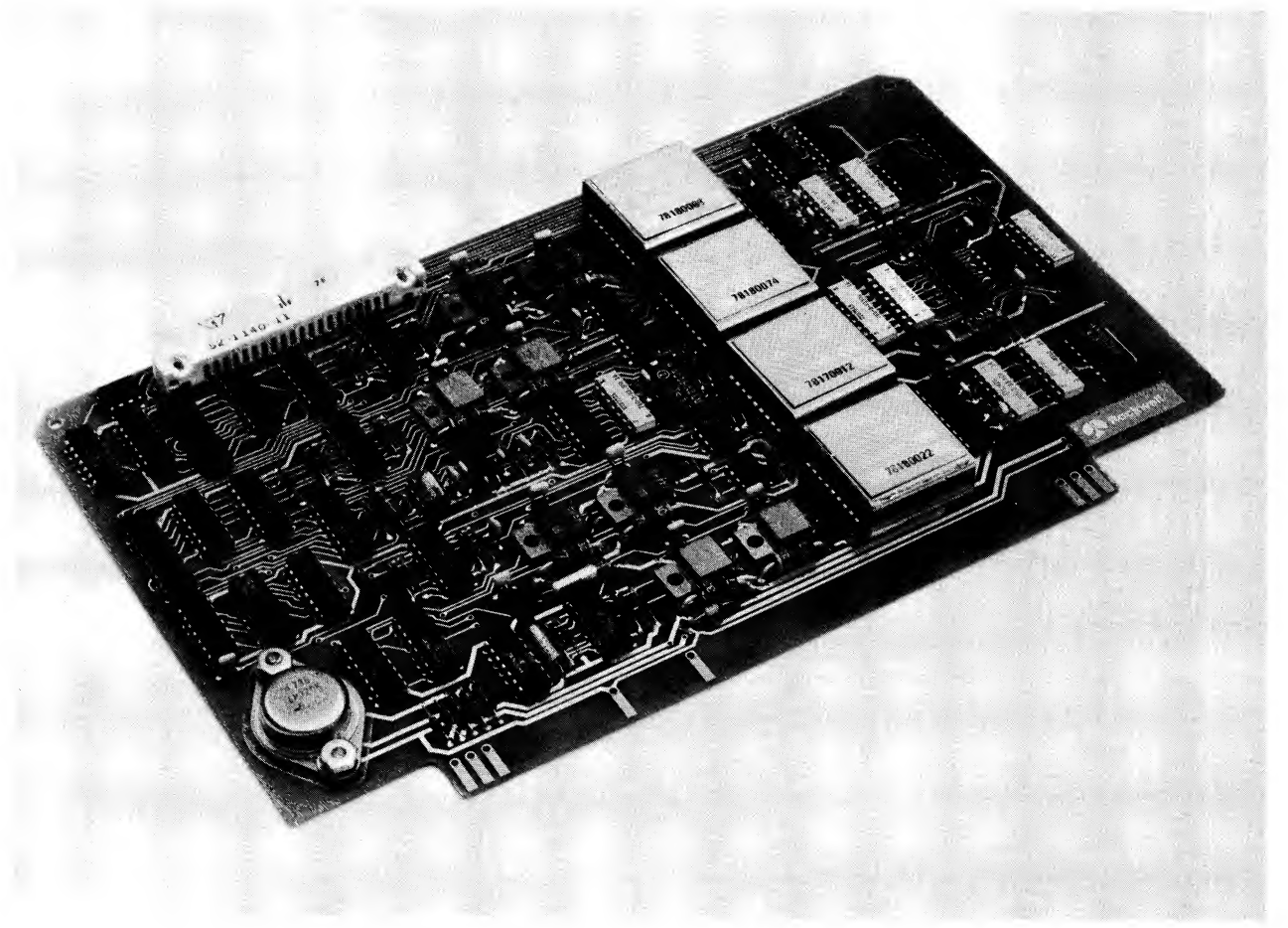


FIGURE 8.27 Rockwell 1-megabit linear bubble memory module. (Courtesy Rockwell International.)

the memory location. Then after a delay of 209 run cycles as shown in Figure 8.29*b*, the data entry begins by pulling the R/WF line to ground.

The next signal shown on the timing diagram is the TIF or transfer in signal. This signal when energized transfers a block of data from the write track into the storage loop. The GENF signal is a timing signal used to generate a bubble. It is noted from the figure that the generate pulse is present at this time and the data are valid on the four data input lines. A total of 282 cycles are used to represent the 282 minor loops of data, each of which contains 1025 bit positions, making a total of 289,050 bit locations. Note that only 260 of the 282 loops are used, leaving 22 of the loops redundant. In actual practice, after manufacture and test of a bubble memory device, the 22 loops with the most marginal performance are labeled "defective" and are not used, thereby increasing the processing yield.

The ROSF signal is a replicate out stretch signal, and the ROCF is a replicate out cut; both are used for replicating operations.

Finally, PROMF is a PROM enable signal used to output loop information.

Intel 7110. Another imporant bubble memory system is the Intel 7110. The Intel 7110 is a 1-megabit bubble memory device implemented on a 2-cm^2 chip, shown in Figure 8.30, that uses bubbles 2.7 microns in diameter. The 7110 is intended to be implemented in a system including the following support circuits:

- 7220 bubble memory controller.
- 7230 current pulse generator.
- 7242.
- 7250 coil.

The entire system is designed to interface with the Intel 8-bit microprocessors, such as the 8080, 8085, and 8088.

The various commercial bubble memory devices presently available or announced are summarized in Table 8.4.

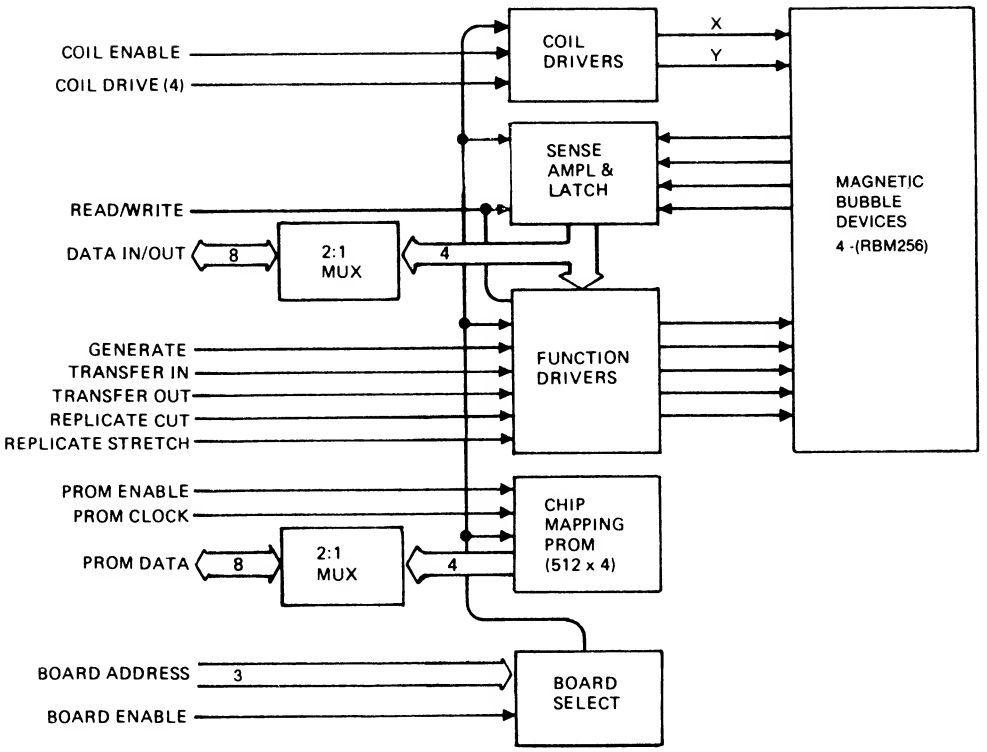


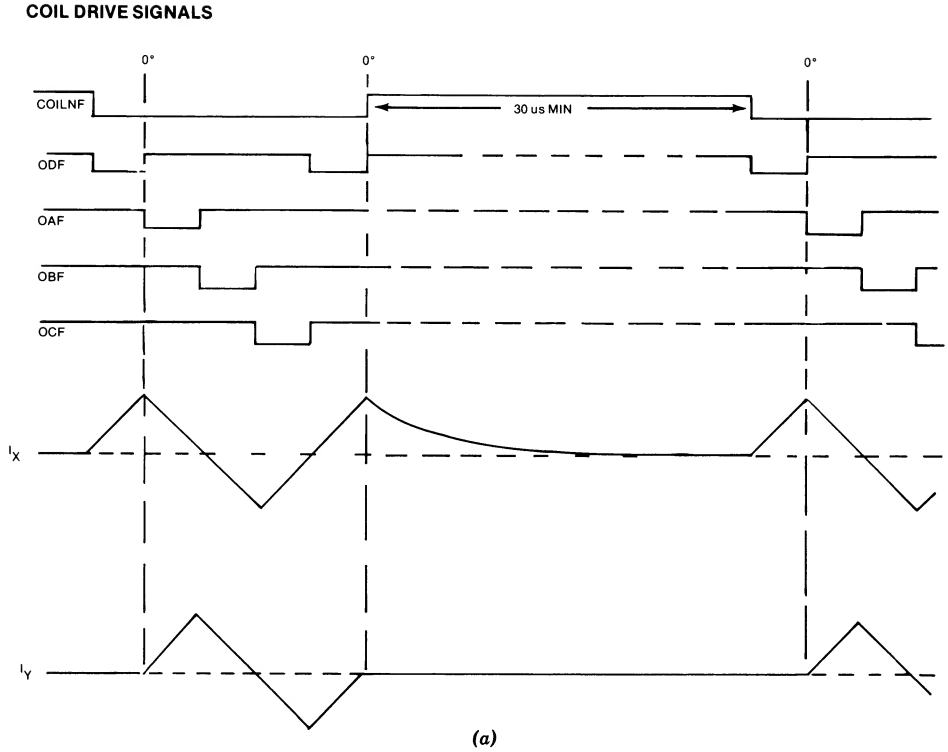
FIGURE 8.28 Rockwell RLM658 module block diagram. (Courtesy Rockwell International.)

Charge Transfer Devices. "Charge transfer device" (CTD) is by now the well-known generic name for devices that store and transfer charge carriers in a storage medium by means of appropriate potentials applied to series of electrodes disposed on an insulating layer overlying one surface of the medium. These devices may be of the charge-coupled (CCD) or bucket brigade (BBD) type.

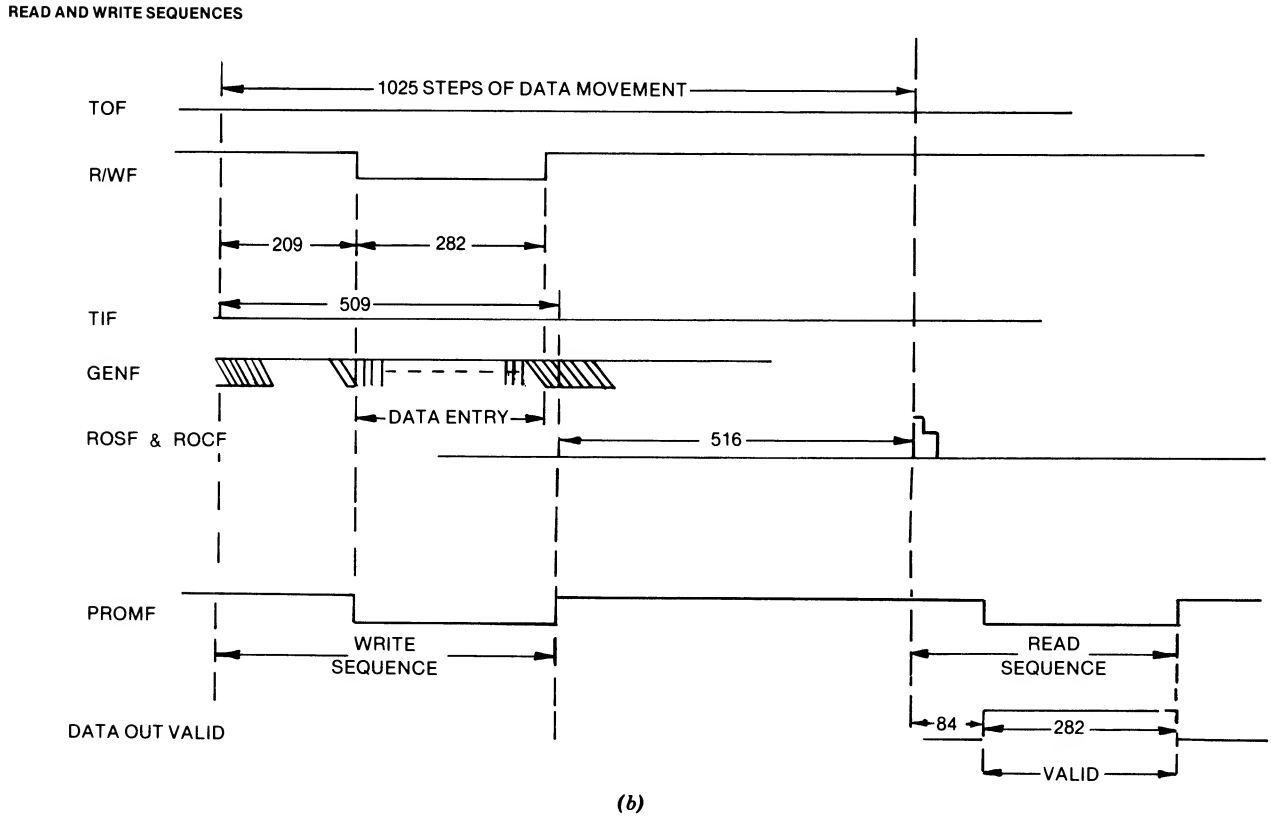
In either category, a spatially periodic electrode metallization pattern on a major surface of a semiconductor body coated with oxide defines a sequence of integrated MOS (metal-oxide-semiconductor) type capacitors. Localized electrical charge accumulations (or "charge packets") formed in the semiconductor, originally in response to an input signal, can be shifted through the semiconductor body sequentially between the adjacent

Table 8.4. Magnetic Bubble Memory Devices

Manufacturer	*Device*	*Capacity*	*Organization*	*Package*	*Cost ($) (unit quantity)*	*Introduced*
Fujitsu	FBM0102	64 Kbits	Major/minor	18-pin	100	Early 1979
	FBM0201	64 Kbits	Serial/parallel	18-pin	100	Early 1979
	FBM0301	256 Kbits	Maj/min block rep	16-pin	500	Early 1979
Intel	7110	1 Mbit	Maj/min block rep	Leadless	2000	Late 1979
National Semiconductor	NBM2256	256 Kbits	Maj/min block rep	18-pin	—	Late 1979
Rockwell	RBM256	256 Kbits	Maj/min block rep	18-pin	500	Late 1978
	—	1 Mbit	Maj/min block rep	18-pin	—	—
Texas Instruments	TIB0203	92 Kbits	Major/minor	14-pin	100	1978
	TIB0303	254 Kbits	Maj/min block rep	20-pin	500	Spring 1979
	—	1 Mbit	Maj/min block rep	18-pin	—	—



(a)



(b)

FIGURE 8.29 Rockwell RLM658: (a) *coil drive signals;* (b) *read and write sequences. (Courtesy Rockwell International.)*

FIGURE 8.30 Intel 7110 1-megabit bubble memory chip. (Courtesy Intel.)

MOS capacitors by a sequence of clocked electrical voltage pulses applied to the electrodes. Signal charges are initially injected at the input end of a chain of such MOS capacitors, in accordance with a stream of digital or analog information, for example, in the form of signal controlled injected charges into a first transfer site of the CTD at the appropriate moments of the clocked voltage pulse sequence.

In the basic bucket brigade device, regions of fixed charge are provided in the storage medium beneath each electrode and extending slightly into the area below an adjacent electrode in the charge transfer path. When an electrode is pulsed, the region of charge immediately under it is reverse-biased and the channel between this region and its neighbor is inverted to permit the transfer of charge. Thus mobile charge carriers are stored in fixed charge regions as majority carriers and transferred through the channels as minority carriers. The basic charge-coupled device stores charge carriers under depletion-biased electrodes and transfers the charge carriers by creating a succession of potential wells at the storage medium surface along the transfer path. Charge is therefore stored and transferred in the form of discrete packets of minority carriers in the medium.

A charged-coupled device (CCD) is a charge transfer device that operates by moving quantities of electrical charge in a controlled manner across the surface of a semiconductor body as a result of a proper sequence of clock pulses. The CCD was in fact discovered during an examination of electrical analogs to magnetic bubble devices.

The two most important types of charge-coupled devices are surface channel CCDs and bulk channel or peristaltic CCDs. Surface channel CCDs transfer and store the signal charge in a potential well at the surface of the semiconductor body at the interface of silicon and silicon dioxide. The devices are considered "charge coupled" because the depletion regions on the silicon surface overlap so that the potential wells merge or couple. An electrical consequence of such coupling is that any mobile minority charge carriers accumulate in the region of the highest interface potential.

In the peristaltic CCD an epitaxial or ion implanted silicon layer is employed so that the maximum potential does not lie at the silicon dioxide interface but in the bulk silicon itself. The mobile minority carriers then reside at the potential maximum in the interior of the silicon. The transfer electrodes then function to modulate the channel potential for changing the level of the potential wells.

It would be worthwhile to illustrate an example of a CCD device at this point. Figure 8.31 is a highly simplified diagrammatic view of a three-phase CCD imager along with the corresponding substrate surface potential during the integration time. In this example, the CCD imager includes a *p*-type substrate with a plurality of electrodes insulated from and capacitively coupled to the substrate. The illustration is diagrammatic in the sense that the insulation employed is not specifically shown and the electrodes are only shown schematically. For an actual three-phase device, the electrodes may overlap one another or may be side to side. In the actual imager configuration, the channel shown in Figure 8.31 is only a portion of one vertical channel of the image sensing region of the array.

During the operation of the CCD imager, known as the "integration" time, radiation is focused onto the top

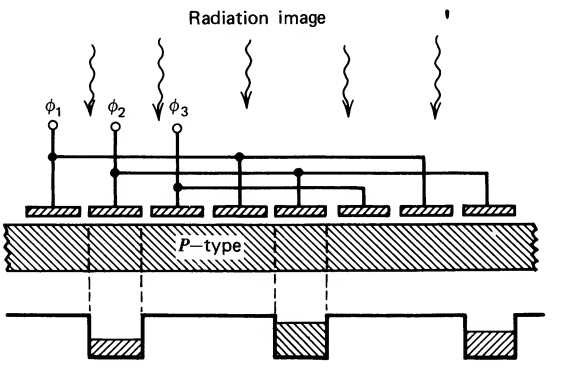


FIGURE 8.31 Three-phase charge-coupled device (CCD) imager.

surface of the array and one of the electrodes in each stage (for example, ϕ_2 electrodes) is maintained at a fixed voltage level. The fixed voltage level of ϕ_2 causes a depletion region for a potential well to form beneath such electrodes, as shown by the heavy dark line representing the substrate surface potential profile in Figure 8.31. The ϕ_1 and ϕ_3 electrodes are maintained at potentials so that the substrate regions beneath such electrodes are in mild depletion or in slight accumulation. The heavily depleted regions are regions from which majority carriers (holes) have been repelled and where minority carrier surface charge (electrons) may accumulate. The lightly depleted (or mildly accumulated) region of the substrate form potential barriers between the potential wells.

As radiation is applied to the CCD cells, charges accumulate in the various potential wells dependent on the radiation intensity reaching the substrate in the region close to the respective ϕ_2 electrodes. As shown in Figure 8.31, the radiation intensity in the region of the middle ϕ_2 electrode is greater than that in the region of the ϕ_2 electrode to the extreme left or to the extreme right. Therefore more charge carriers accumulate in the middle potential well than in either the left hand or right hand potential well. In the present example the charge carriers are electrons.

It is important to understand how the imager is actually operated and reference may be made to Figure 8.32, which shows the timing wave forms employed in the device of Figure 8.31. During the integration time the ϕ_2 voltage is at a fixed level so that the depletion regions are formed. The ϕ_1 and ϕ_3 electrodes are also maintained at a fixed level but below the voltage at which depletion occurs, so that the electrodes are in mild accumulation only. Operation in such a manner is intended to control "blooming." Blooming is defined as the spreading of the charge originally accumulated in a light sensing element in such a way as to interact with charge accumulated in adjacent light sensing elements.

As is indicated, the integration time may be relatively long, for example, in the case of commercial television under 1/60-sec. Upon termination of the integration time interval, the charge is stored in the imager and transferred to a so-called B register (a temporary storage array) and in turn from the B register one row at a time to a C register. The transfer of the charge from the A register to the B register is accomplished by the multiple phase wave form that may last some 900 μsec.

For some time there has been interest in the CCD device for use as a memory. There are three basic types of CCD memory organization:

- Synchronous.
- Serial-parallel-serial.
- Line addressable.

The synchronous CCD organization, shown in Figure 8.33, includes a plurality of shift registers arranged in a serpentine fashion. Refresh amplifiers are provided between each CCD shift register. Such an organization is fast but uses a considerable amount of power.

The serial-parallel-serial organization shown in Figure 8.34, as the name implies, uses a parallel array of CCD shift registers having their outputs connected serially. Such an approach is relatively slow but utilizes relatively low power.

The line addressable organization shown in Figure 8.35, basically uses a parallel array of CCD shift registers in which each line of data may be accessed at random through the address selection matrix. Such an approach is the fastest in operation and dissipates relatively low power.

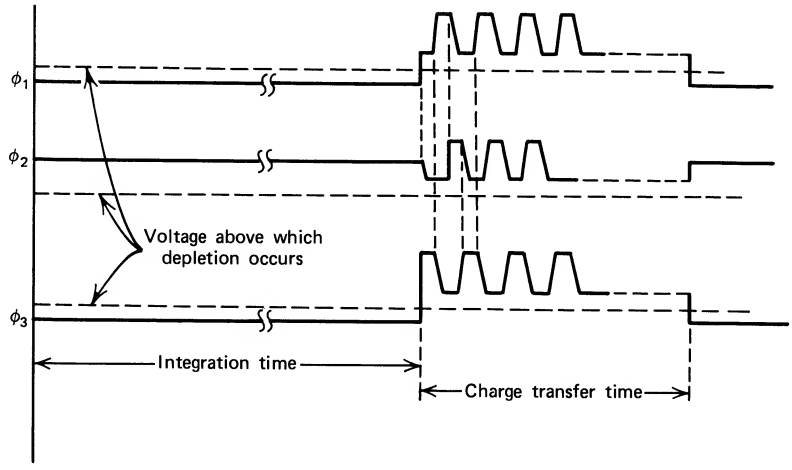


FIGURE 8.32 Timing wave forms of three-phase CCD imager.

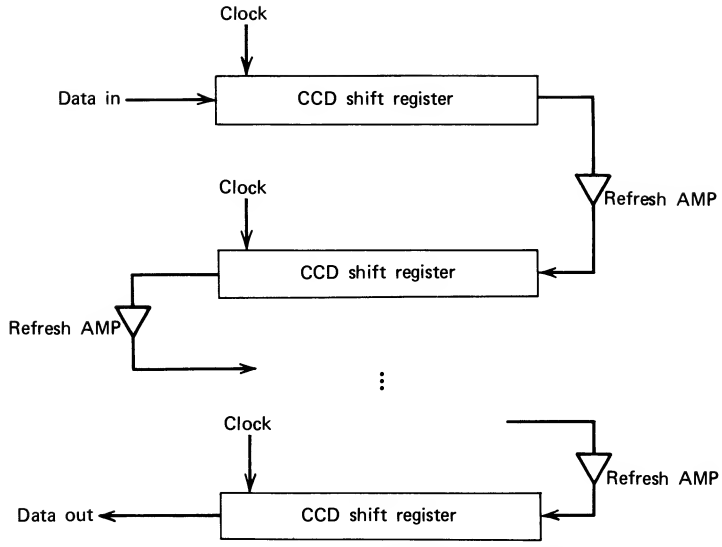


FIGURE 8.33 Synchronous CCD memory organization.

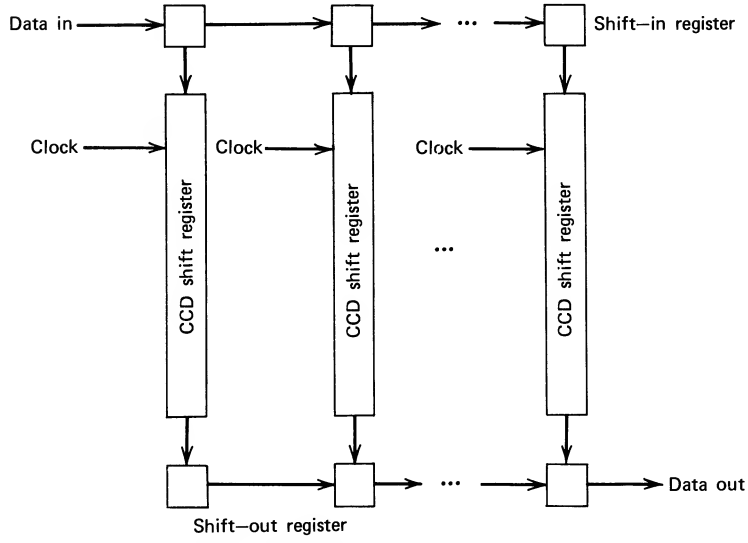


FIGURE 8.34 Serial-parallel CCD memory organization.

Examples of the serial-parallel-serial type of CCD memory include the Intel 16K 2416, while an example of the line addressable CCD is the Fairchild 16K 460. The features of the various types of CCD memory organizations are summarized in Table 8.5.

However, because of the rapid development of RAM designs in MOS technology, CCD memories are presently not competitive on either a price or performance basis with MOS technology.

Table 8.5. CCD Memory Organization

	Serpentine	*Serial-Parallel-Serial*	*Line-Addressable*
Capacity	16K	16K	16K
Typical organization	256 X 16 X 4	4096 X 4	128 X 32 X 4
Access Time	50 μsec	820 μsec	25 μsec

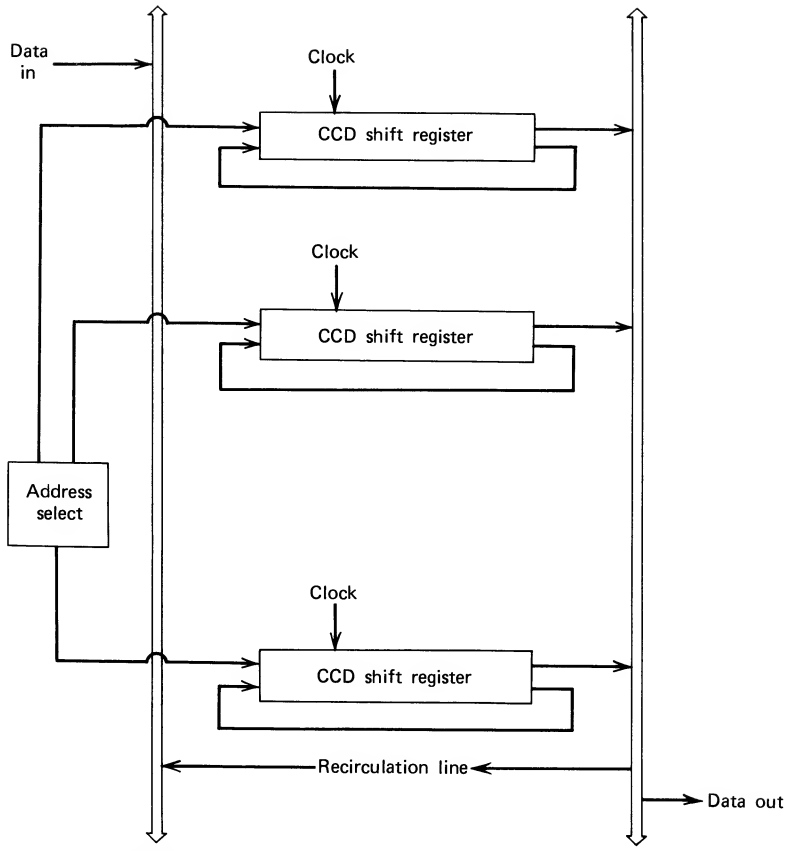


FIGURE 8.35 Line addressable CCD memory organization.

I/O Interface Control

I/O interface control is concerned with the means for controlling the connection between the processor and memory and the external devices that provide information to the system or are controlled by the system.

There are three types of interfacing control:

- Programmed I/O.
- Memory mapped I/O.
- Direct memory access I/O.

Programmed I/O refers to the use of explicit program instructions in the instruction set of the microprocessor for performing I/O functions. In terms of hardware, programmed I/O refers to specific I/O ports on the microprocessor that are controlled by such I/O instructions, as represented in Figure 8.36. All the data transfer between the microprocessor and the peripheral device over the data bus takes place in response to signals on these I/O lines and under explicit instruction control of the microprocessor.

The instructions associated with programmed I/O

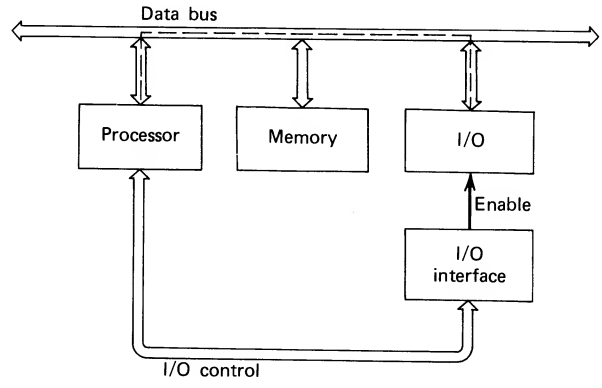


FIGURE 8.36 Programmed I/O or I/O mapped I/O.

are typically IN and OUT instructions. In the 8086, the IN and OUT instructions are 1 or 2 byte instructions. The 2-byte instruction is a fixed port instruction, such as:

```
IN 0
OUT 255
```

where the byte following the IN or OUT operand is an integer from 0 through 255 that designates in which one of the 256 possible ports the I/O operation is to take place. The 1-byte instruction,

IN (DX)
OUT (DX)

assumes that the port number is found in the DX register. Since the DX register is a 16-bit register, this instruction permits up to 64K input or output ports.

Memory mapped I/O refers to the use of memory for transferring data to and from a peripheral device. More specifically, memory addresses are used in place of peripheral addresses, and the peripherals "replace" some of the normal memory of the system. Memory access instructions are automatically decoded by the peripheral devices, and when a specific peripheral device is being addressed it then interfaces directly with the system data bus. The microprocessor does not "recognize" that a peripheral device has taken the place of a memory location and performs data read and write to that peripheral device as if it was a memory location.

A simplified representation of memory mapped I/O is shown in Figure 8.37. The principal advantage of memory mapped I/O over programmed I/O is that the system design has all the memory reference instructions available for performing I/O operations. Thus greater flexibility in programming I/O operations is possible with memory mapped I/O. The principal disadvantages are:

- I/O operations are directly performed by the CPU, therefore postponing other processing tasks until the I/O is complete.
- Memory addresses must be set aside for peripheral device use, therefore decreasing the actual memory address workspace of the system.

The third type of interfacing control is direct memory

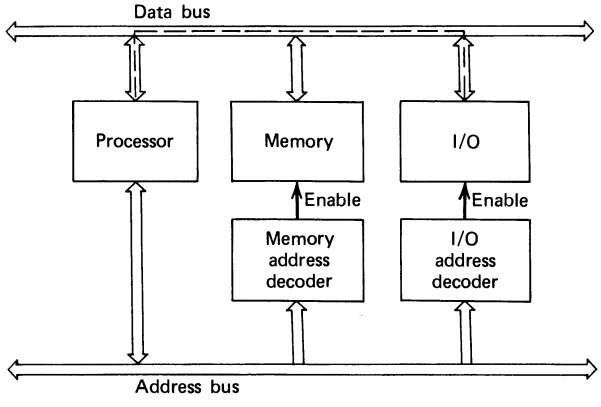


FIGURE 8.37 Memory mapped I/O.

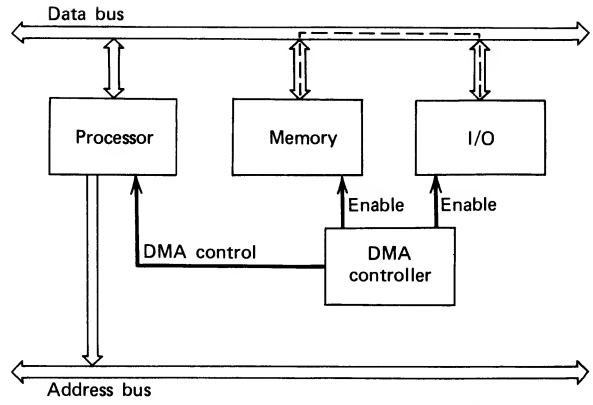


FIGURE 8.38 Direct memory access (DMA).

access (DMA) I/O. DMA is a technique that permits direct access between the system memory and the peripheral device without CPU intervention. A simplified representation of a DMA system is shown in Figure 8.38.

In programmed I/O, the data bus carries the data to be transferred, while the address bus carries the designation of the peripheral device. External logic must be used to decode the M/IO signal on the microprocessor to distinguish between a memory and an I/O bus cycle. A typical implementation of programmed memory mapped I/O is shown in Figure 8.39 using the 8086 microprocessor. Use is made of a 1 out of 8 binary decoder (the 8205) and a programmable peripheral interface (the 8255).

When the 8205 is enabled, one of its eight outputs goes low, which is connected to the active low chip select input of a peripheral device or peripheral interface. The 8205 is enabled when both the E_1 and E_2 enable inputs are low.

In the 8086, the $\overline{BHE}$ and A_0 signals are used to select the appropriate bytes of the memory or I/O word to be read or written, and therefore in addition to being connected to the M/IO line, the 8205 in Figure 8.39 is connected to either the $\overline{BHE}$ or the A_0 line. The appropriate selection is made on the basis of the following data:

$\overline{BHE}$	A_0	
0	0	Whole word
0	1	Upper byte from/to odd address
1	0	Lower byte from/to even address
1	1	None

Turning again to Figure 8.39 if BHE is high (i.e., 1), and A_0 is low (i.e., 0), then we know that the processor wants to tell the interface that the lower byte on the

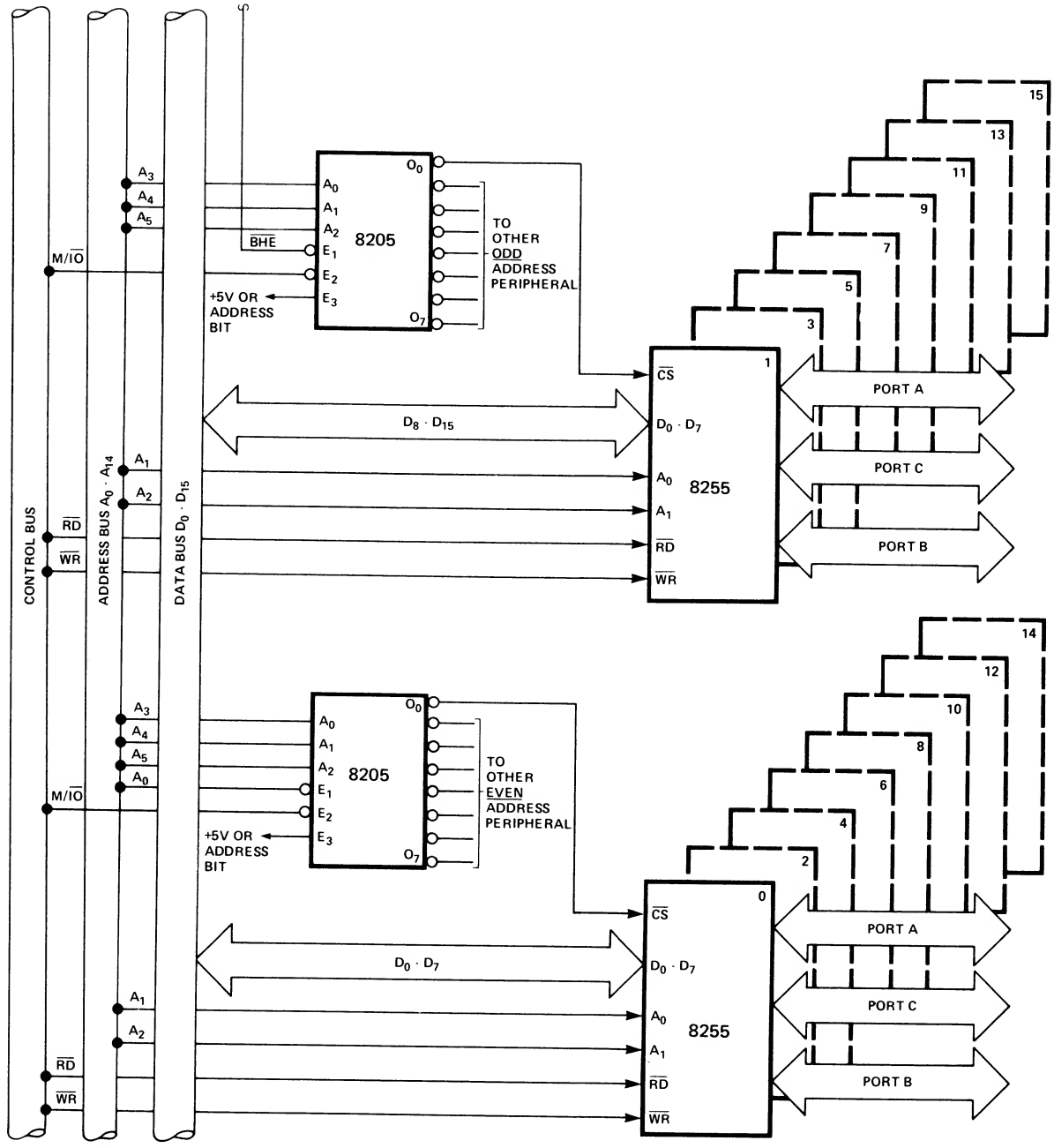


FIGURE 8.39 Programmed I/O with the 8086. (Courtesy Intel Corporation.)

data bus is to be transferred from or to an even addressed I/O port. Since BHE is high, the top 8205 in Figure 8.39 is not turned on, but the bottom 8205 is turned on. We further assume that A_3, A_4, and A_5 are low, so that the output O_0 of the bottom 8205 is enabled. The output O_0 is connected to the chip select line of the lower 8255, which is then enabled.

To contrast memory mapped I/O with programmed I/O, an 8086 implementation of memory mapped I/O is presented in Figure 8.40.

At this point, it would be worthwhile to review the features of the Intel 8255 programmable peripheral interface in greater detail.

The 8255 Programmable Peripheral Interface

The Intel 8255A is a general-purpose programmable I/O device for use with microprocessor systems. It is "programmable" in the sense that system software, in the form of a control word sent to the 8255A, is able

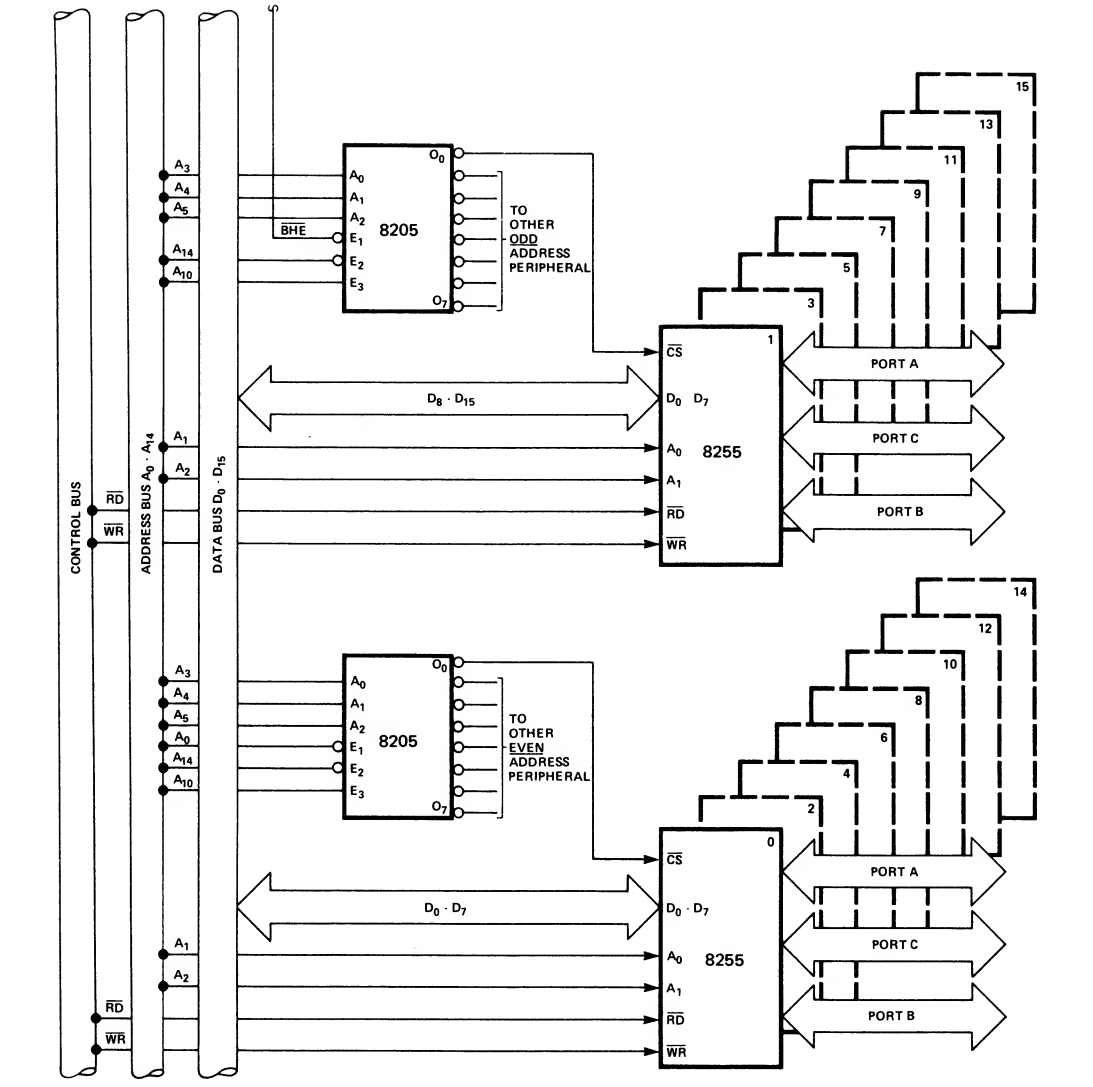


FIGURE 8.40 Memory mapped I/O using the Intel 8086. (Courtesy Intel Corporation.)

to set the 8255A to a specific operating mode to perform the required interfacing task.

The basic block diagram of the 8255A is shown in Figure 8.41. It is noted that the 8255A interfaces with the microprocessor system through a bidirectional data bus and is controlled by means of a read/write control logic block that has control inputs:

CS: Chip select

RD: Read

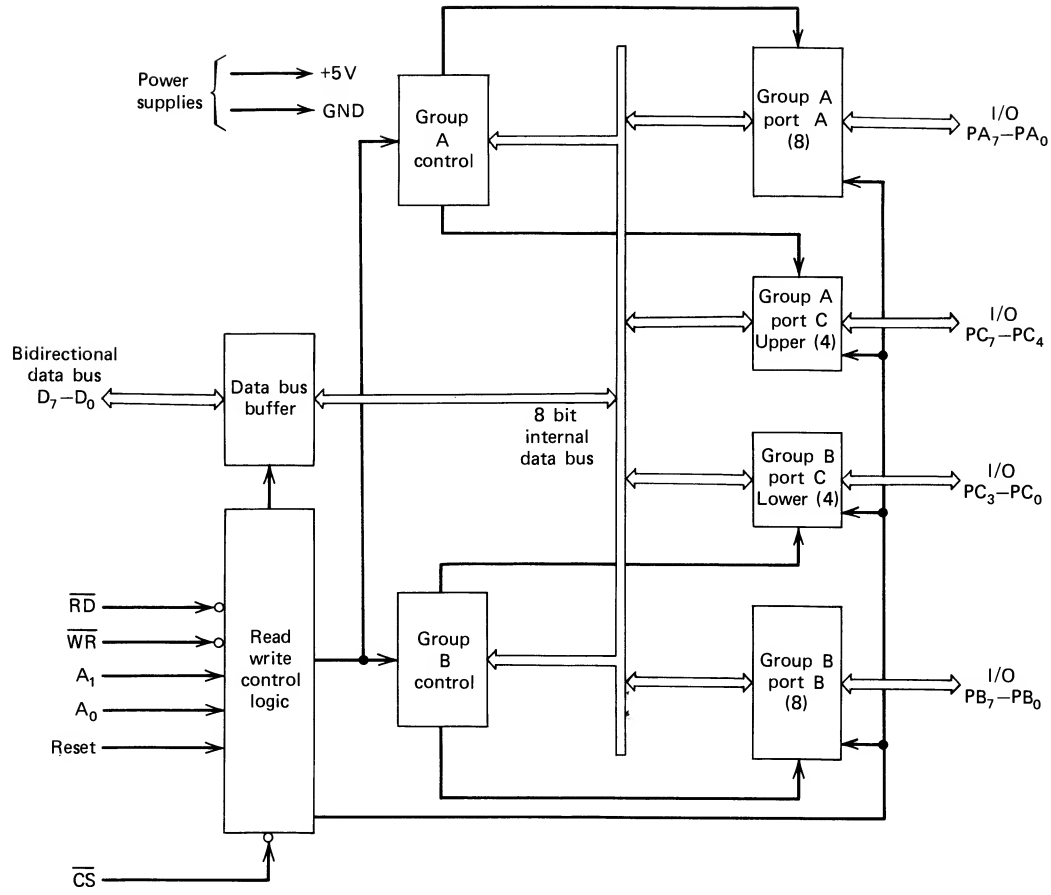


FIGURE 8.41 Intel 8255A block diagram. (Courtesy Intel Corporation.)

WR: Write

A_0: Port select 0

A_1: Port select 1

RESET

The I/O ports of the 8255A are divided into three 8-bit groups (A, B, and C) that can be configured by system software by means of an 8-bit control word. The various modes and their relationship with the control word are shown in Figure 8.42.

Mode 0 is a basic input/output mode that provides I/O for each of the three ports without handshaking. Mode 1 is a strobed I/O in which handshaking is provided. Finally, mode 2 forms a strobed bidirectional I/O bus.

It would be worthwhile to illustrate a few of the possible interfacing applications using the 8255A. Figure 8.43 shows a digital/analog and analog/digital converter interface, while Figure 8.44 is a floppy disk interface.

CRT Interfacing

The CRT display is one of the most important output devices for a microcomputer system, particularly for small business and word processing systems. Almost all CRT displays are based on the same scanning method used in a television receiver, and it would therefore be useful to begin a discussion of CRT interfacing with a review of the scanning technique of a television receiver.

"The CRT display for a television receiver in the United States is composed of 525 lines that are scanned 30 times per second, such as shown in diagrammatic form in Figure 8.45. An electron beam starts at the upper left hand corner and scans lines from left to right; at the end of each line the beam rapidly retraces to the left hand side of the screen, during which time a blanking circuit turns off the beam. After the beam reaches the bottom right hand corner, it rapidly jumps back to the upper left hand corner, a blanking circuit again preventing any display. The video signal supplied to the CRT display is composed of horizontal and vertical synchronization pulses that activate such blanking circuits at appropriate intervals. The sequence of black and white images in each of the scanned lines is determined by pulses that are situated between the horizontal sync pulses, such as shown in Figure 8.46. Figure 8.46 also shows the signal timing.

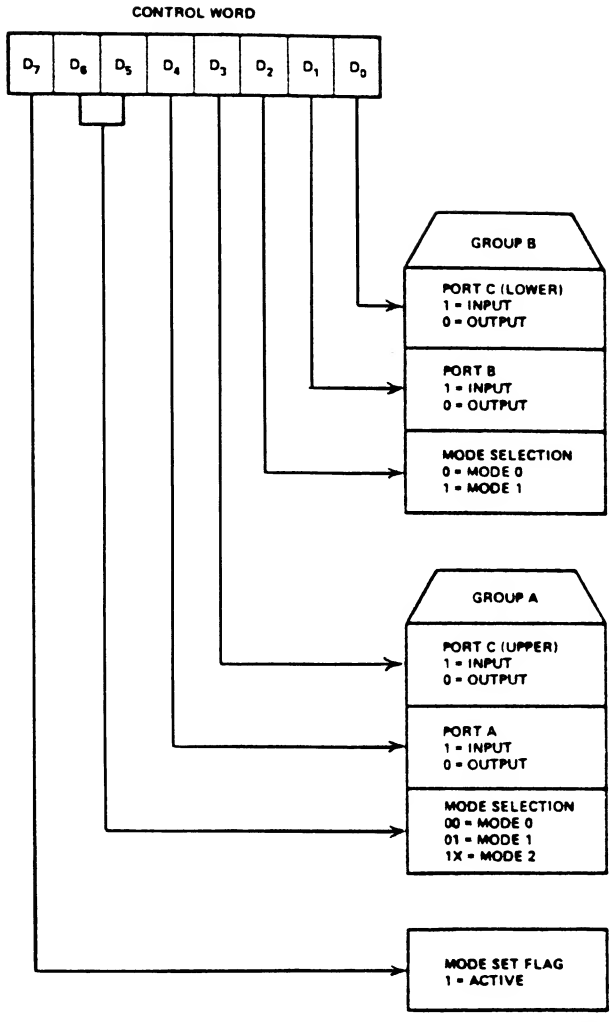


FIGURE 8.42 Intel 8255A modes and control word. (Courtesy Intel Corporation.)

"The scanning pattern and the video signal timing are different for broadcasts in many other countries, and thus division receivers are also different electronically. In Europe, a standard of 625 lines that are scanned 25 times per second is used."*

Now that the scanning process is understood, the next logical question is how characters are formed from the dots on the CRT display.

Character Generation

"Character generation on a CRT display is implemented by breaking each character into a dot matrix pattern and storing such dot matrix information in a ROM. When a particular character is to be displayed, the address of the

*Adapted from Daniel R. McGlynn, *Personal Computing* (Wiley, New York, 1979).

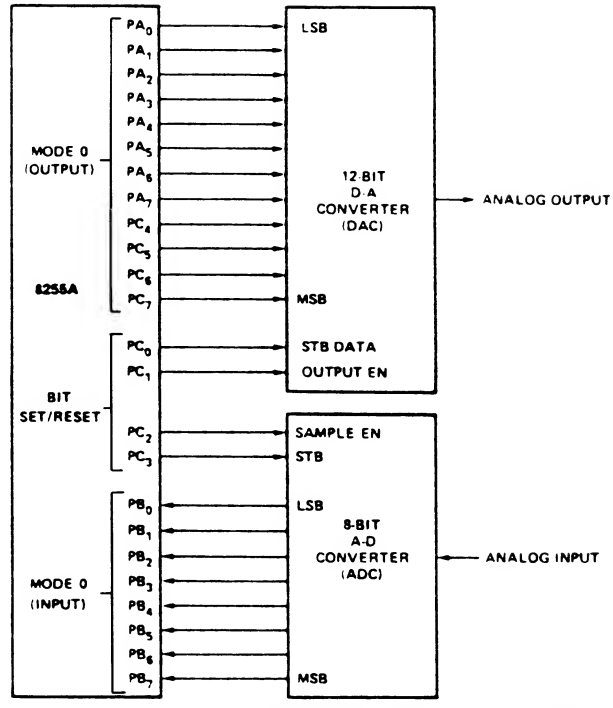


FIGURE 8.43 D/A and A/D interface with the 8255A. (Courtesy Intel Corporation.)

character is applied to the ROM, and the ROM provides the appropriate dot matrix pattern to be displayed.

"The representation of the character in ROM is usually in the form of a 5 X 7 or a 7 X 9 dot matrix. Figure 8.47 is a 5 X 7 dot matrix representation of the ASCII character set in a commercially available character generator ROM.

"The size of the dot-matrix pattern for the characters limits the number of characters per line and the lines of characters that can be displayed on the standard television receiver. Allowing space between character lines, up to 10 scanning lines on the display screen are used for representing one line of characters. The typical representation on the screen is 16 lines of 32 characters each."*

Video Interface Circuits

The final aspect of video interfacing is the issue of how scanning circuits are controlled so as to produce a predetermined dot matrix character on the CRT display. One of the most important developments in this area has been the development of monolithic integrated circuits for CRT interfacing. In the present section we consider the Standard Microsystems 5027, shown in functional block diagram form in Figure 8.48.

*Adapted from Daniel R. McGlynn, *Personal Computing* (Wiley, New York, 1979).

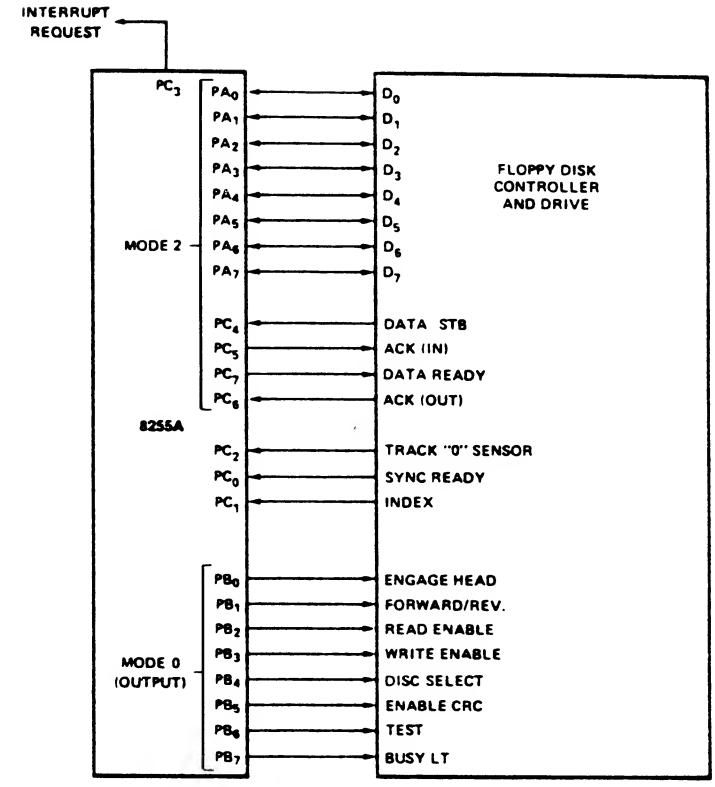


FIGURE 8.44 Floppy disk interface with the 8255A. (Courtesy Intel Corporation.)

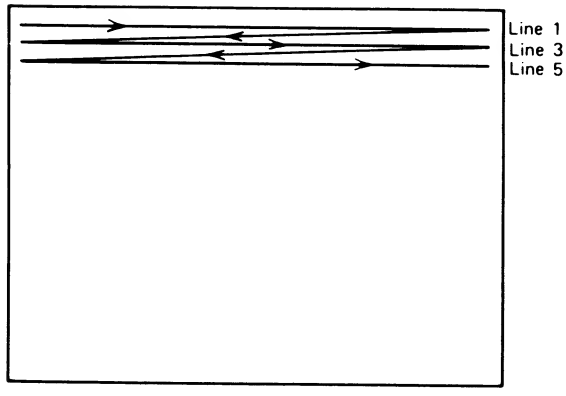


FIGURE 8.45 CRT scanning pattern.

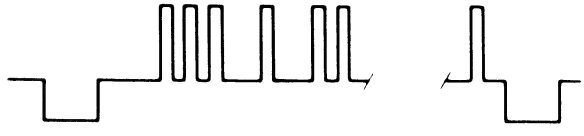


FIGURE 8.46 CRT signal timing.

The 5027 video timer control (VTAC) device is used in a circuit such as that shown in Figure 8.49 for interfacing the microprocessor, the dot counter, and the character generator to the CRT. The dot counter, which is clocked at a frequency above 25 MHz, is a discrete component since such high clock rates cannot be presently implemented in an MOS circuit like the 5027.

The functions of the 5027 VTAC are:

- Frame formatting.
- Horizontal, vertical, and composite synchronization.
- Specification of characters per data row.
- Specification of data rows per frame.
- Specification of raster scans per data row and per frame.

These features are selected by the system designer according to the particular display requirements of the intended application. The 5027 simplifies the design task by offering the designer a variety of standard formats and allowing him to select the most appropriate by programming the 5027 with a control code. The 5027 is user programmable through the use of seven 8-bit control registers shown in Figure 8.50. In addition to the control registers there are two other registers in Figure 8.50 that are related to cursor operations.

A description of the registers and their functions, as taken from in Standard Microsystems literature, is given below.

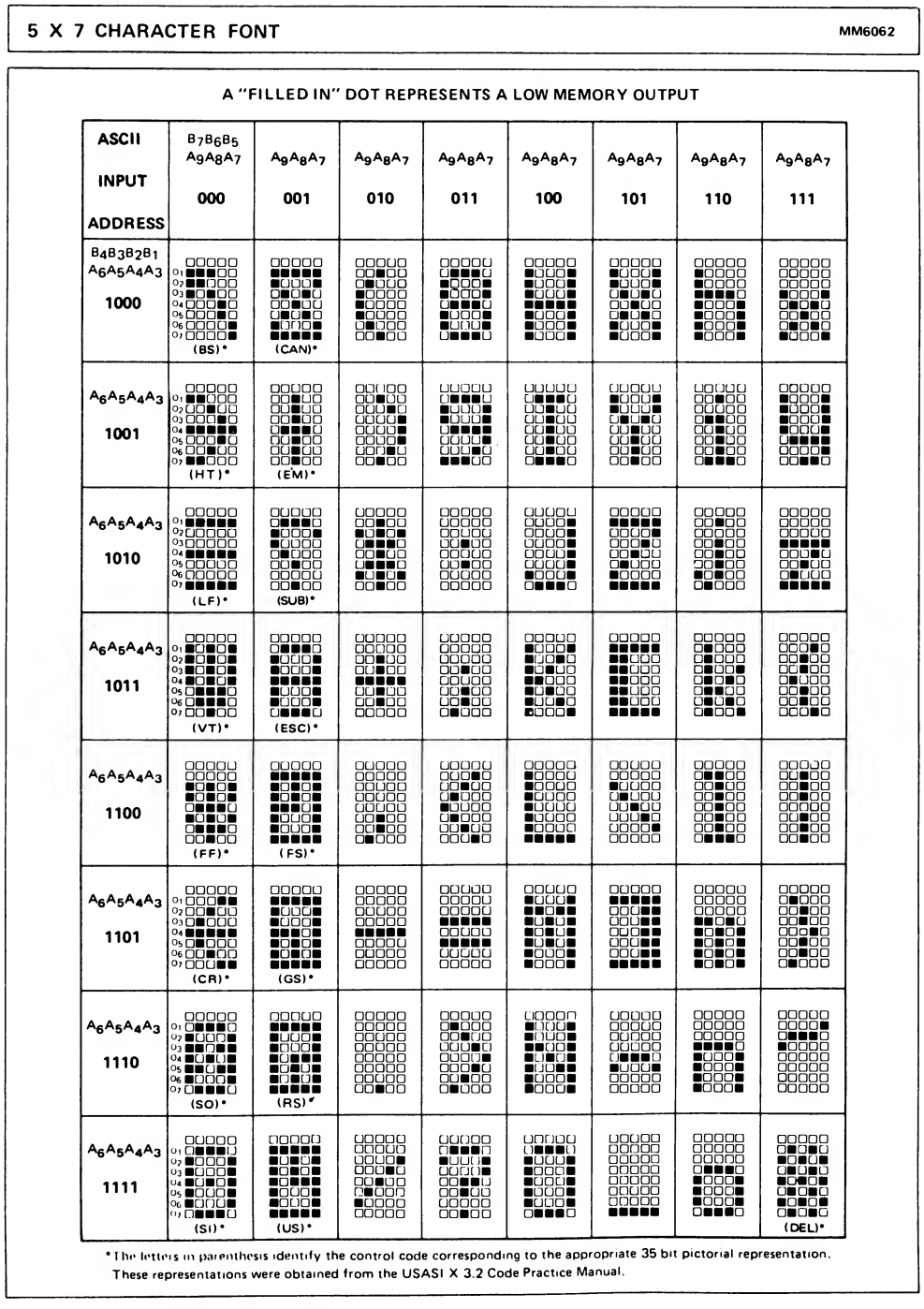


FIGURE 8.47 Characters in ROM, 5 X 7 dot matrix representation.

"*Register 0.* This 8-bit register contains the number of character times for 1 horizontal period of the TV raster scan. For example, using American Standard Television (63.5 μsec per line) at a character time of 500 nsec, the value for this register would be 63.5 divided by 0.5 = 127. The number in this register is normally 1.25 times the number of characters per line displayed on this screen. The value loaded into this register is the binary equivalent of 126 (127 - 1). Since character times are counted from zero instead of one, the value loaded into this register is one less than the actual number of character times.

"*Register 1.* This register contains 3 fields of information. The most significant bit (7) is the interlace bit. If

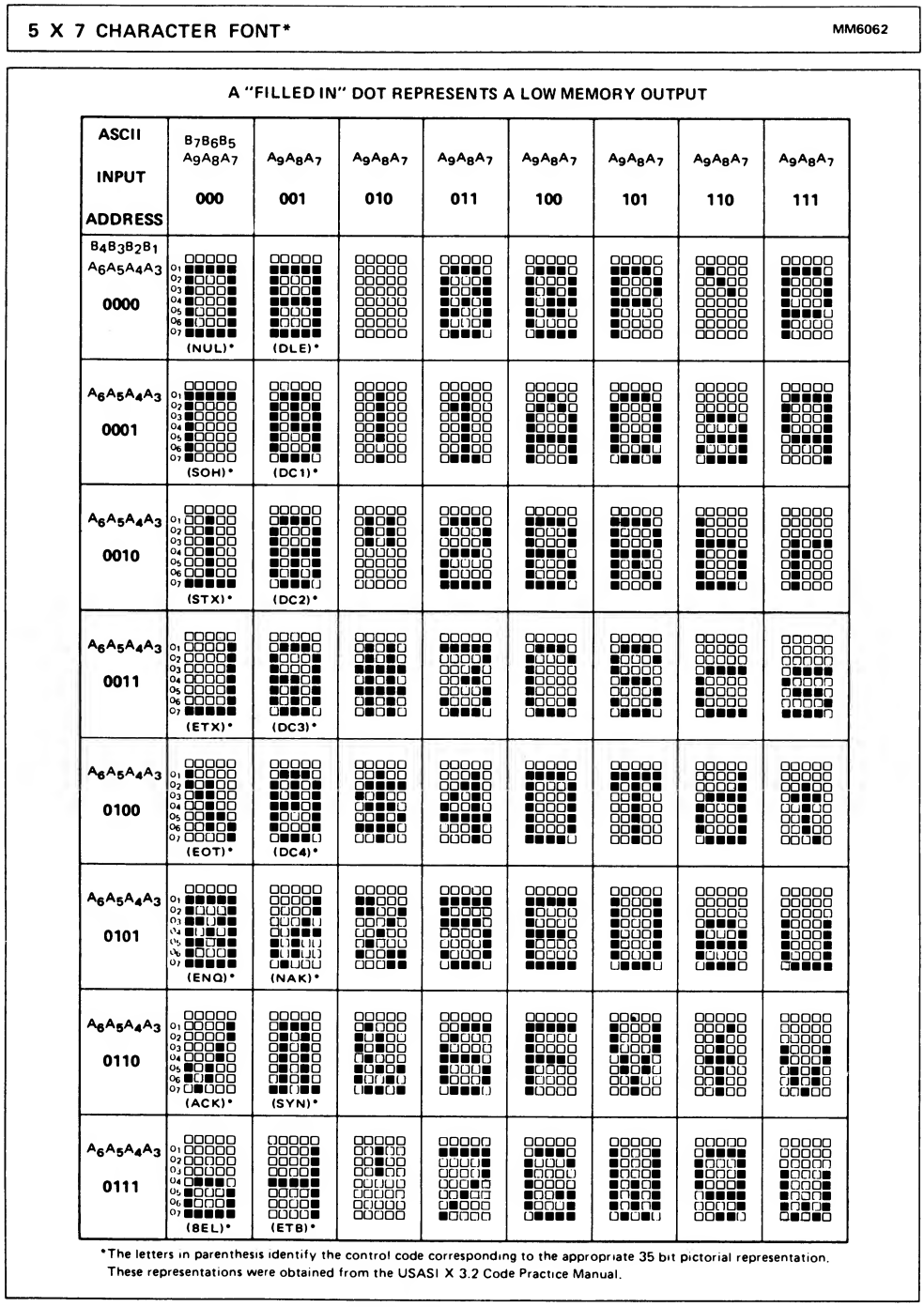


FIGURE 8.47 (Continued)

this bit is set to a 1, Interlace mode is indicated; if set to a 0, Noninterlace mode is indicated. The next 4 bits (6-3) define the number of character times for the width of the horizontal sync pulse. For example, using American Standard Television (4.5 μsec) and a character time of 500 nsec indicates that it would require 9 character times; therefore the binary equivalent 9 would be loaded in these bits. The least significant 3 bits (2-0) are used to specify the horizontal sync delay. This is commonly called the Front Porch and is the period between the end of active video to the beginning of the horizontal sync pulse. The value here is not critical and can be used to position the video horizontally on the screen.

Block Diagram

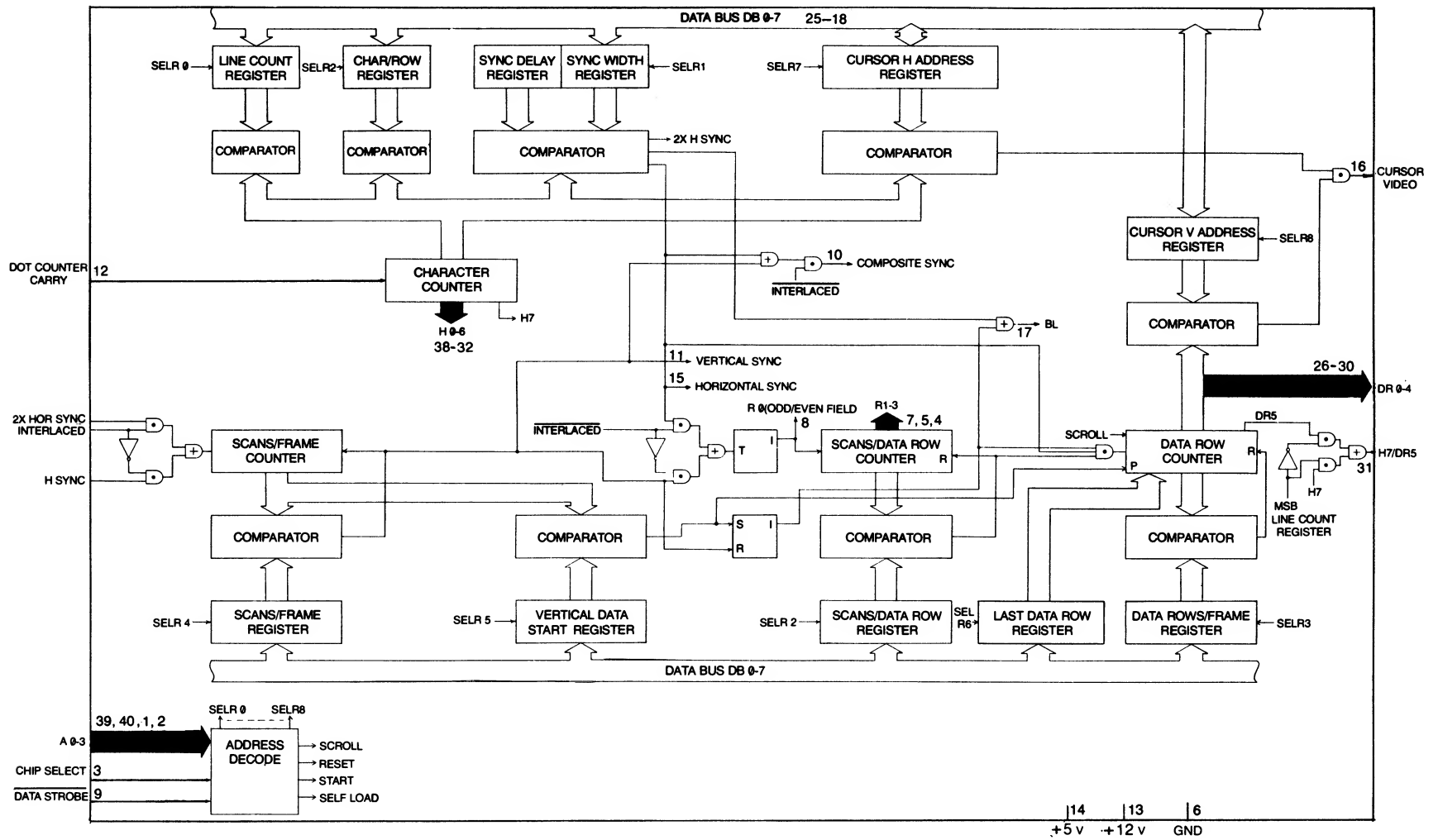


FIGURE 8.48 Standard Microsystems 5027 functional block diagram. (Courtesy: Standard Microsystems.)

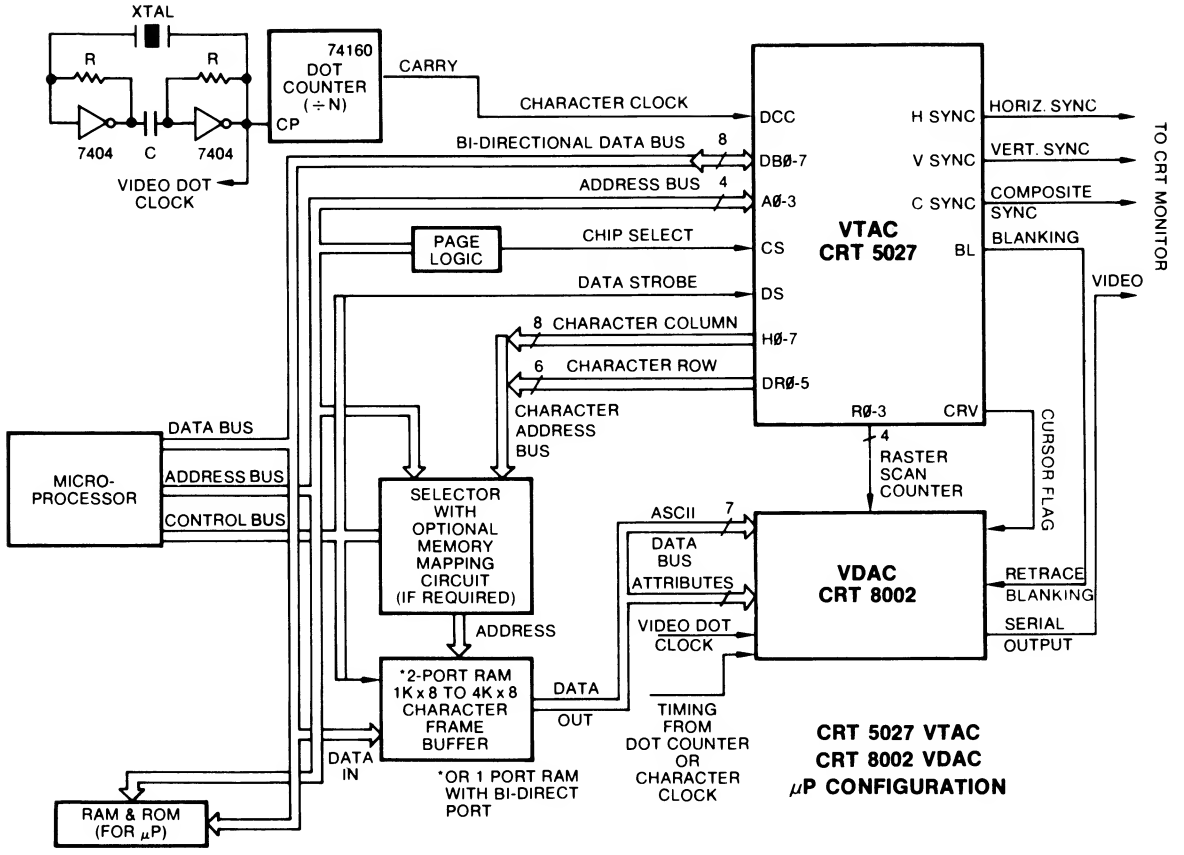


FIGURE 8.49 Video interface circuit using the 5027. (Courtesy Standard Microsystems.)

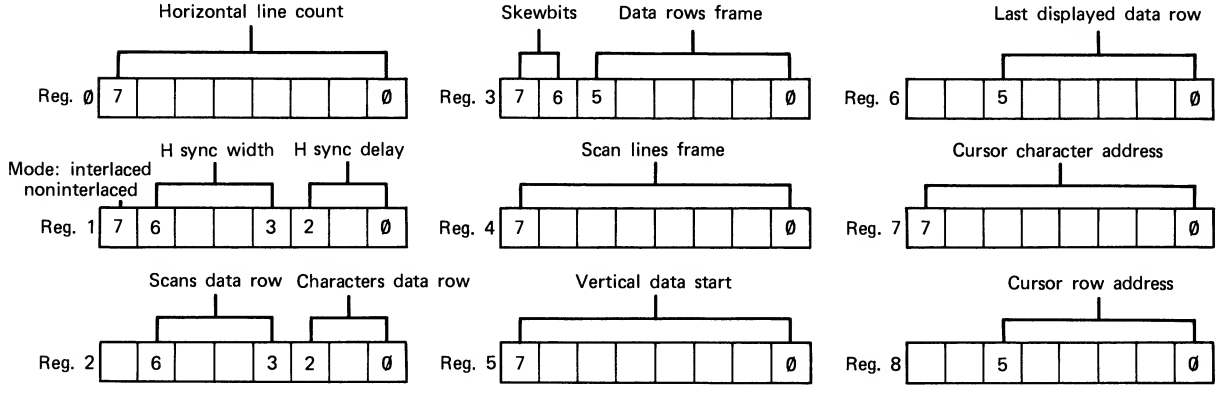


FIGURE 8.50 5027 control registers. (Courtesy Standard Microsystems.)

"*Register 2.* This register contains both the number of characters to be displayed per line and the number of scans per character. Bit 7 is not used (B7 = X). Bits 6 through 3 define the number of scans per character. For example, using a 7 X 9 dot matrix character generator, the normal number of scans might be 12; therefore using 12 scans per character, the binary equivalent of eleven (12-1) is inserted into this field. The least significant 3 bits (2-0) contain a 3-bit code that defines the number of characters per line. The CRT 5027 VTAC is preprogrammed for 20, 32, 40, 64, 72, 80, 96, and 132 characters per line. The 3-bit binary number used in this field determines the particular format, for example, 80 characters being the sixth value would be coded as a binary 5 (101). Any eight formats may be programmed into the CRT 5027 as mask options if nonstandard formats are desired. (see table 8.6)

"*Register 3.* This register contains both the propaga-

Table 8.6. Characters/Data Row

DB2	*DB1*	*DB0*
0	0	0 = 20
0	0	1 = 32
0	1	0 = 40
0	1	1 = 64
1	0	0 = 72
1	0	1 = 80
1	1	0 = 96
1	1	1 = 132

Table 8.7. Skew Bits

		Character Times	
DB7	*DB6*	*Sync/Blank Delay*	*Cursor Delay*
0	0	0	0
1	0	1	0
0	1	2	1
1	1	2	2

tion delay compensation field (skew bits) and the data row fields. Bits 7 and 6 are used to adjust the blanking, cursor position and sync delay so as to compensate for either 0, 1, or 2 character time propagation delays of the character generator and the frame buffer RAM. (see table 8.7) The 6 least significant bits (5-0) define the number of data rows to be displayed on the screen. The number of rows begins at 000000 (single row) and continues to 111111 (64 rows.)

"*Register 4.* This 8-bit register defines the number of raster lines in the field (frame). Care should be taken when programming this register to make sure that the product of the scans per data row times the number of data rows is less than the number of raster scans. There are two methods of programming this register. In the interlaced mode subtract 513 from the number of raster lines desired and divide by 2. For example, for 525 scans, the register should contain the number 6. In the noninterlaced mode subtract the number 256 from the desired number of raster lines and divide by 2. For example, for 262 raster lines, the value is 3.

"*Register 5.* This register defines the number of raster lines between the beginning of the vertical sync pulse and the start of the first data row being displayed. Typically, values of 20 and 21 lines are used. Higher values can be used to position data lower on the screen to a maximum of 255. This is called Vertical Data Start and is the sum of Vertical Sync and Vertical Scan Delay.

"*Register 6.* The least significant 6 bits (5-0) of this register define the last data row to be displayed on the screen. Bits 7 and 6 are not used. This feature is useful for both scrolling and positioning of data. For example, if the display was set for 24 data rows, normally row 0 would be on top of the screen and row 23 would be at the bottom. If the scroll register (register 6) contained the number 15, then row 15 would be at the bottom and row 16 would be at the top of the screen. Row 23 and row 0 would be contiguous in the middle of the screen.

"*Register 7.* This 8-bit register contains the character number at which the cursor is to be addressed. For example, if the last character of an 80 character per line display were to be cursored, the binary equivalent of 79 would be in this register.

"*Register 8.* The least significant 6 bits (5-0) of this register define the data row for the cursor; similar to Register 7."*

Serial Communications Device Interfacing

Serial communications device interfacing refers to interfacing the parallel data from the microprocessor or memory to a serial data channel such as a communications line. There are at least four distinct types of serial communications circuits:

- Universal asynchronous receiver/transmitter (UART).
- Universal synchronous receiver/transmitter (USRT).
- Universal synchronous/asynchronous receiver/transmitter (USART).
- Data link control chips (DLCC).

The function of these circuits is basically to convert the parallel data from the computer to serial form for transmission over a data communication link or other serial channel, and conversely to convert the serial data received over the communication link to parallel data that can be processed by the computer. Hence the name "receiver/transmitter." The specific nature of the communication link (synchronous, asynchronous, etc.) distinguishes among the various types of devices.

The UART is a device that is used with an asynchronous communication link, such as a start/stop control protocol like the teletype. The UART functions to append or delete the start or stop bits, generates and detects the error or parity bit, generates and detects special characters, such as BREAK, and performs automatic resynchronization with each new start bit. Table 8.8,

*Reprinted with permission of Standard Microsystems Corporation.

Table 8.8. UARTs

Harris 6402
Motorola 6850
Western Digital 1620

Table 8.9. USRTs

General Instruments 2350
Motorola 6852
NEC 379

Table 8.10. USARTs

Intel 8251
Western Digital 1671

is a list of commercially available UARTs, and a block diagram of a typical UART is shown in Figure 8.51.

The USRT is a device that is used with a synchronous communication link, that is, a contiguous bit stream with the receive clock locked on the received data. Synchronous communication uses a synchronization character, labeled SYN, that performs such synchronization and is inserted or deleted in the data stream by the USRT as required. Some commonly known USRTs are given in Table 8.9.

The USART is a device that performs the function of both the UART and the USRT. Whether the device acts asynchronously or synchronously is controlled by the user through the use of a control code applied to the chip. Examples of the USART are given in Table 8.10. One example is shown in Figure 8.52, and programmable versions are presented in Figures 8.53 and 8.54.

Finally, the DLCCs are more sophisticated data communications devices that provide support of either bit oriented or byte oriented communications protocols. The devices also function as receiver/transmitters and may operate with asynchronous as well as synchronous communications. The DLCCs perform header generation and detection, bit stuffing or stripping, and cyclic redundancy character (CRC) generation and checking.

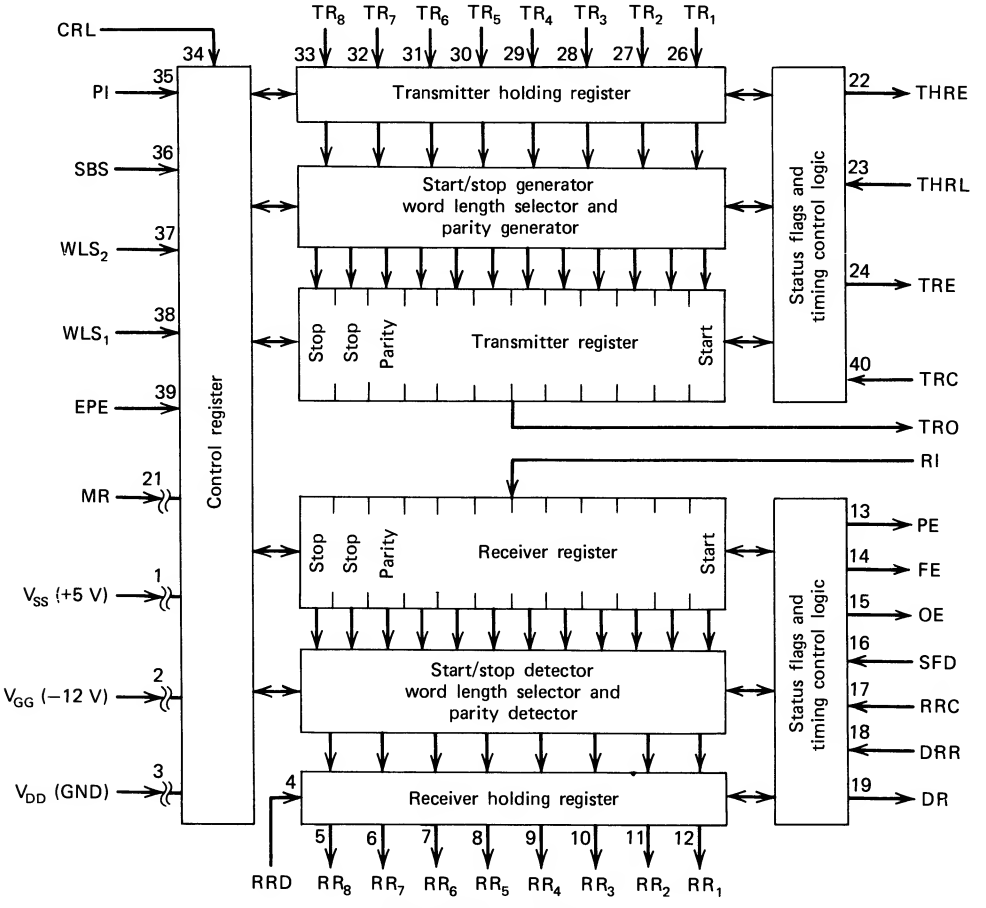


FIGURE 8.51 UART.

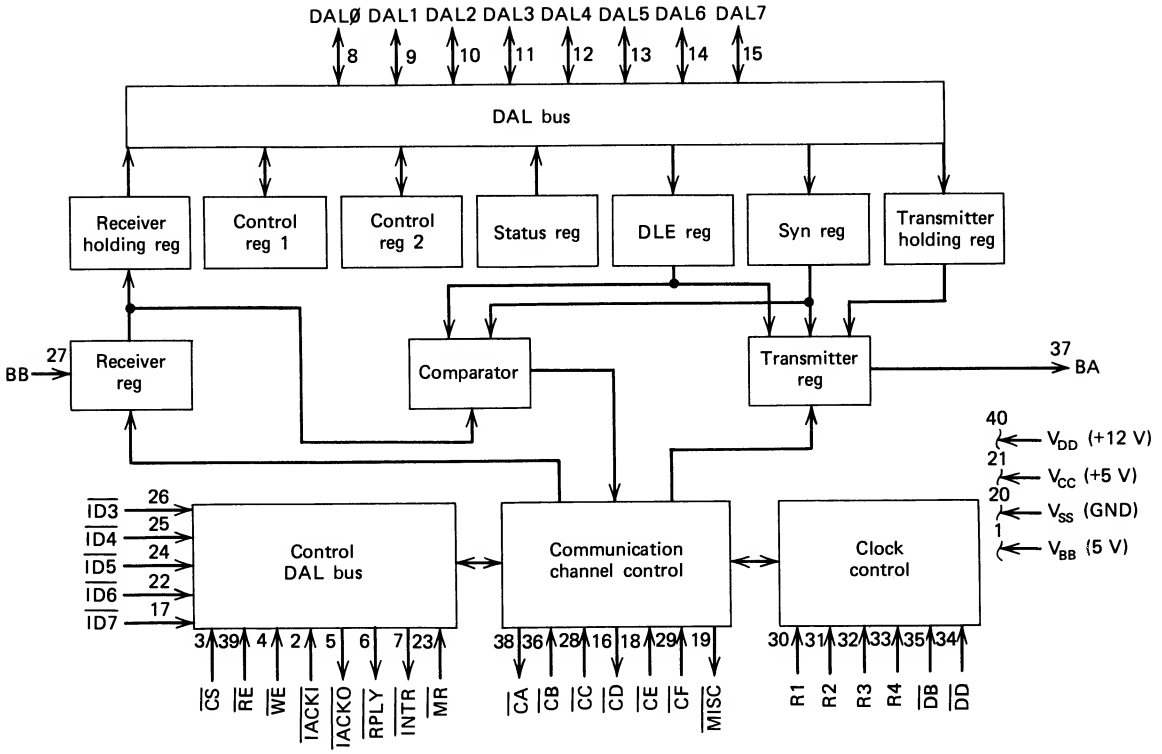


FIGURE 8.52 USART.

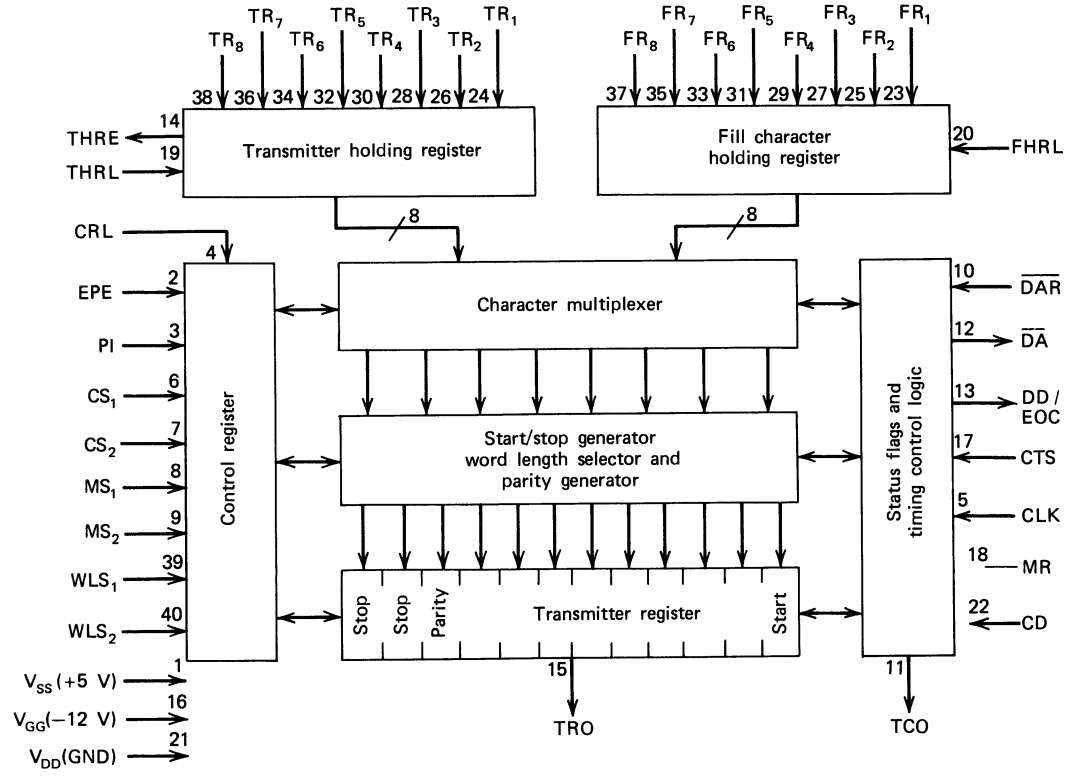


FIGURE 8.53 Programmable USAT.

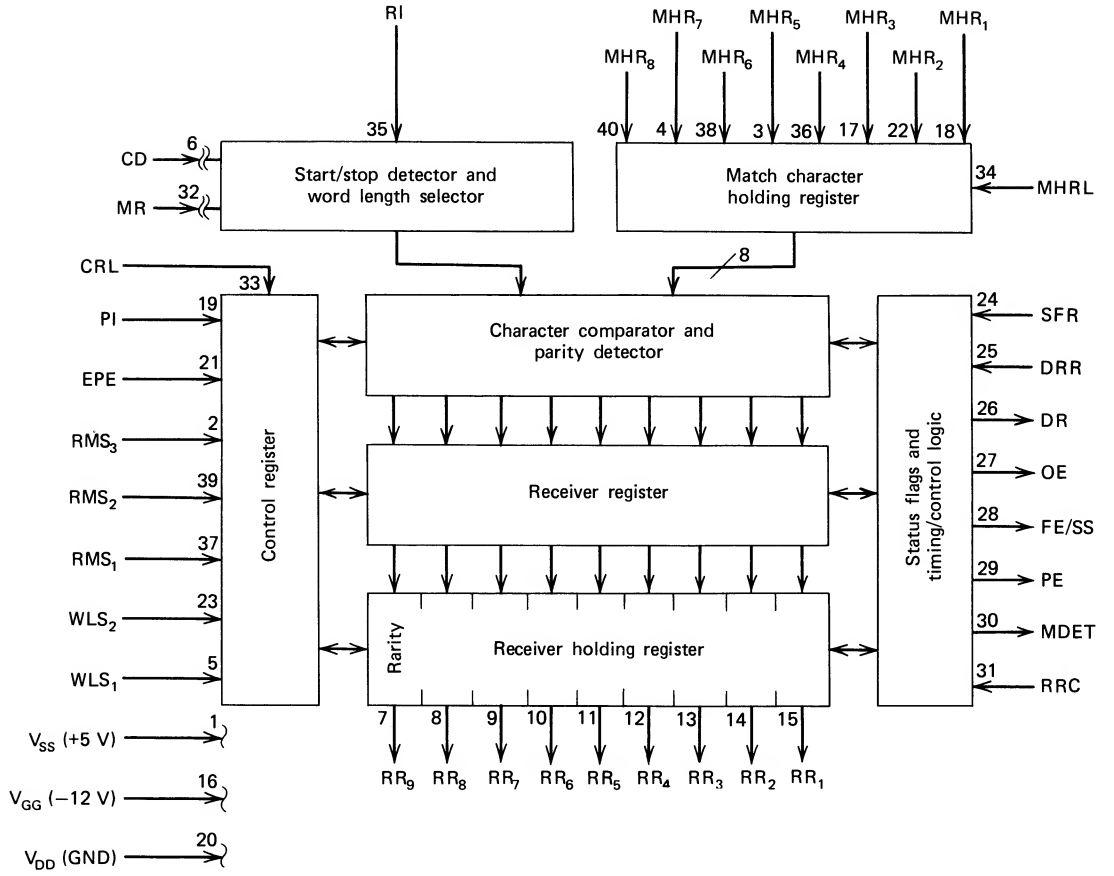


FIGURE 8.54 Programmable USAR.

The DLCCs are described in more detail in the next section.

Multiprotocol Data Link Control Interfacing

As we point out in the preceding section, data link control devices are used to interface a computer system with a data communications network using a particular "protocol" or format. There are numerous protocols in use or proposed, including bit-oriented protocols and byte-oriented protocols.

The most important bit-oriented protocols are IBM's Synchronous Data Link Control (SDLC) and similar protocols expressed by standards such as the CCITT's X.25, and the International Standard Organization's High-level Data Link Control (HDLC).

The most important byte-oriented protocols are IBM's binary synchronous (bi-sync) and Digital Equipment Corporation's DDCMP.

The use of a particular protocol in a data communications network generally depends on the particular computer equipment being used. The problem arises in interfacing a microcomputer system, such as an intelligent terminal, to a data communications network based on a standard protocol. Such interfacing requires various tasks:

- Frame character detection and generation.
- Bit stuffing and stripping (for transparency).
- Error checking.

Such an interfacing task has been greatly simplified through the development of various multiprotocol synchronous receiver transmitters. Standard Microsystem's COM5025, shown in block diagram in Figure 8.55, and Intel's 8273 HDLC/SDLC Protocol Controller are examples of single integrated circuit implementations of such interfaces.

Another example is the Western Digital Micro Packet Network Interface (μPAC) device, an NMOS integrated circuit packaged in a 48-pin package. The μPAC is a bit-oriented protocol interface device dedicated to the level 2 control of the CCITT X.25 protocol. Some of the features found on the μPAC not found on the general-

Block Diagram

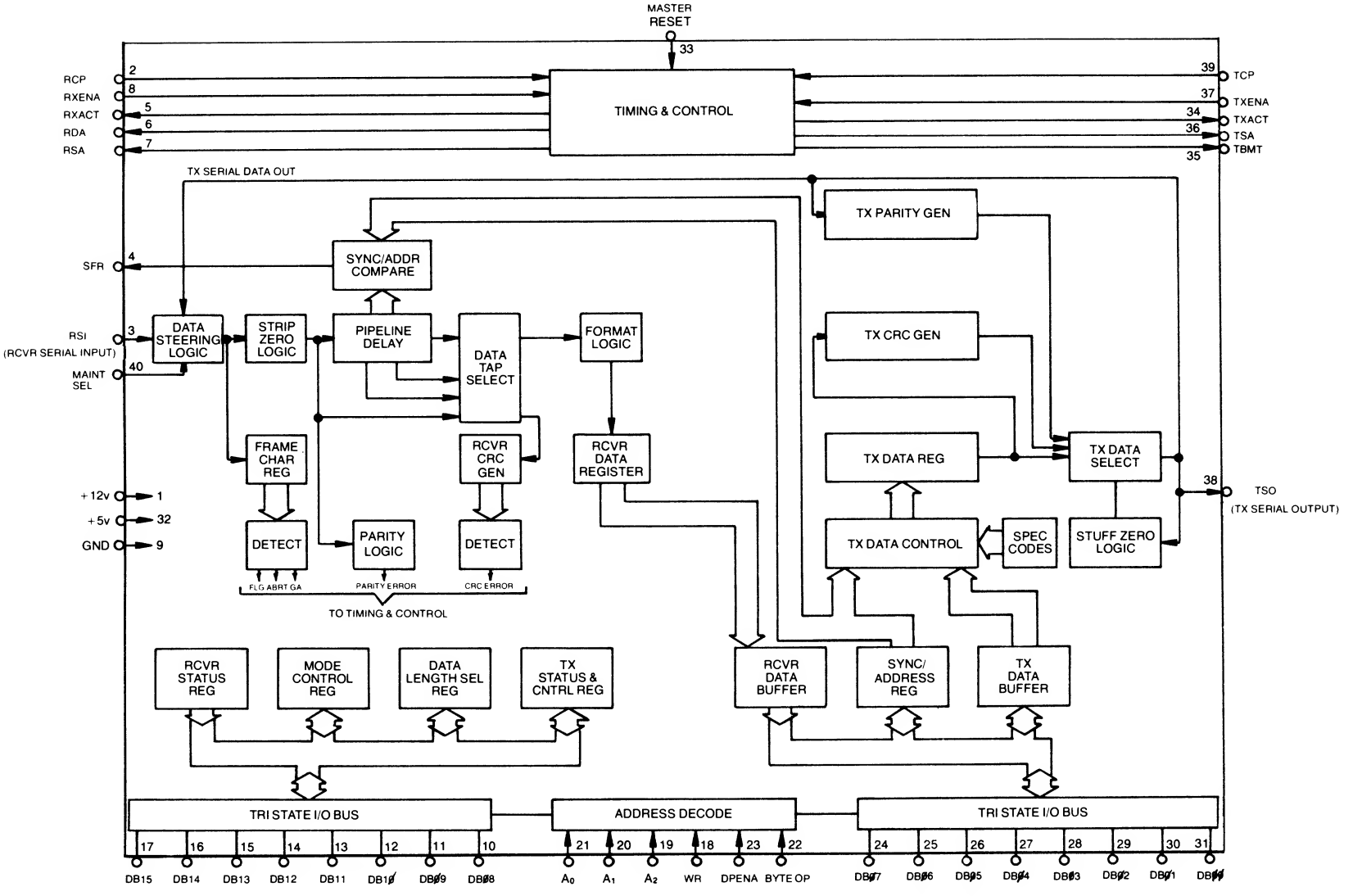


FIGURE 8.55 Standard Microsystems 5025 functional block diagram. (Courtesy Standard Microsystems.)

purpose bit-protocol devices include retransmission facility, link set-up, disconnect, and reset, time-out recovery, and multipoint operation capability.

Encrypted Data Link Interfacing

Encrypted data link interfacing refers to interfacing a computer system with a data link that handles encrypted data for security reasons. The use of encrypted data transmission is important in many banking and financial applications, including potential applications in electronics funds transfer, where secure data is a major requirement.

One of the standard data encryption techniques is the Federal Information Processing "Data Encryption Standard" (DES). The DES uses a 56-bit user-specified key to produce 64-bit cipher words.

Floating Point Processors

Another very important peripheral circuit in microprocessor systems used particularly in computational applications is the so-called "floating point," or arithmetic processing unit. Such devices are actually dedicated, special-purpose microprocessors that are specifically designed for handling high speed fixed and floating point arithmetic and trigonometric operations. Since such processors do not have the logical and memory address capabilities of general purpose microprocessors, their role is to support the arithmetic processing capability of the general purpose microprocessor used in the system.

The first such device, the Advanced Micro Devices 9511, was introduced in 1978 and inspired other semiconductor manufacturers to consider offering similar products, such as the Intel 8087 designed for use with the 8086 microprocessor. In 1979 AMD itself announced a follow-on arithmetic processing unit, the 9512.

AMD 9511

The AMD 9511 is a *n*-channel silicon gate MOS processor designed for performing fixed point 16- and 32-bit operations. The functional block diagram of the 9511 is given in Figure 8.56 and the pin diagram is shown in Figure 8.57.

As seen from the pin diagram, the 9511 has an 8-bit data bus, and the instruction format is an 8-bit byte shown in Table 8.11. The instruction format specifies the following features:

- Specific operation (bits 0–4).
- Data format (bits 5–6).
- Service request (bit 7).

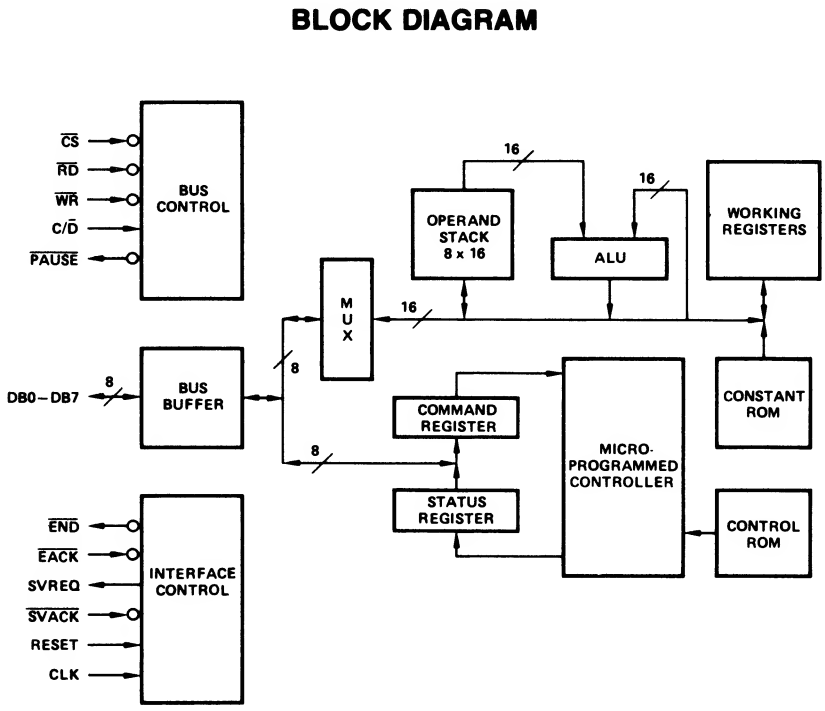
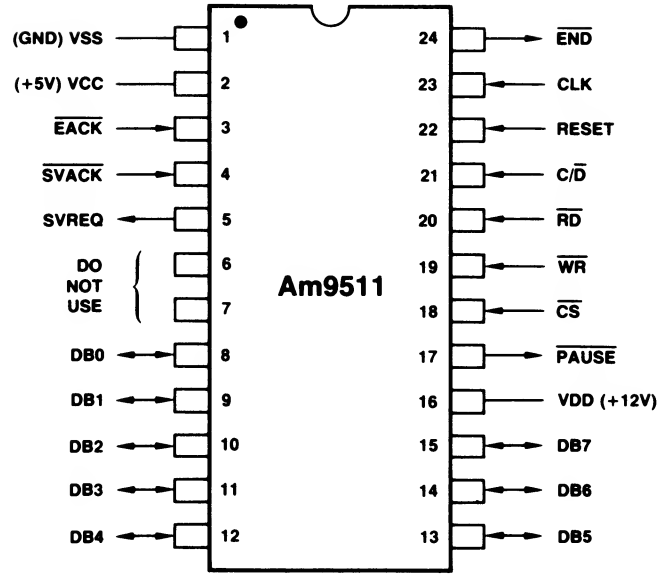


FIGURE 8.56 AMD 9511 functional block diagram.

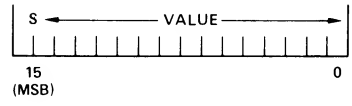
CONNECTION DIAGRAM
Top View



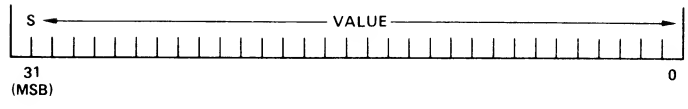
Pin 1 is marked for orientation.

FIGURE 8.57 AMD 9511 pin diagram.

16-BIT FIXED POINT FORMAT



32-BIT FIXED POINT FORMAT



FLOATING POINT FORMAT

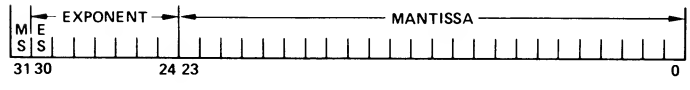


FIGURE 8.58 AMD 9511 data formats.

There are five types of commands or instructions for the 9511:

- Fixed point 16-bit.
- Fixed point 32-bit.
- Floating point 32-bit.
- Derived floating point functions.
- Data manipulation commands.

If bit 5 of the instruction is a 1, a fixed point data format is specified. If bit 5 is a 0, a floating point data format is specified. Bit 6 specifies the precision of the data for fixed point instructions: 1 for single precision (16-bit), 0 for double precision (32-bit). The actual commands or instructions for the 9511 are listed in Table 8.11. The data formats for fixed point and floating point operation are illustrated in Figure 8.58.

Interfacing the 9511 to a microprocessor is suggested in Figures 8.59 and 8.60. Figure 8.59 shows an example of a minimum configuration, while Figure 8.60 shows an example in which transfers are handled by a DMA controller, in this case the Advanced Micro Devices 9517.

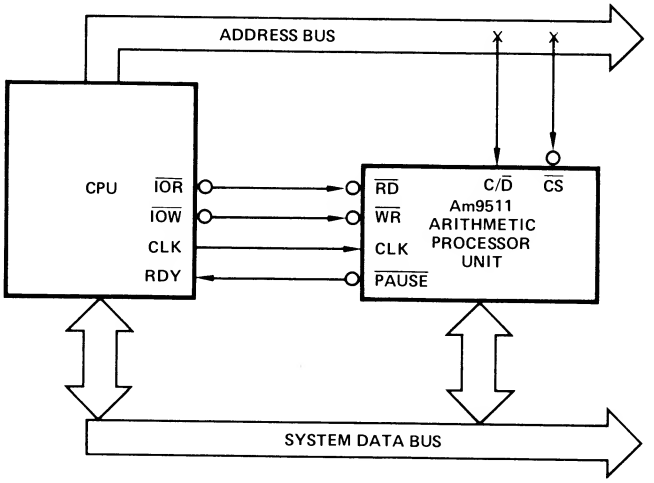


FIGURE 8.59 AMD 9511 interfacing: minimum configuration.

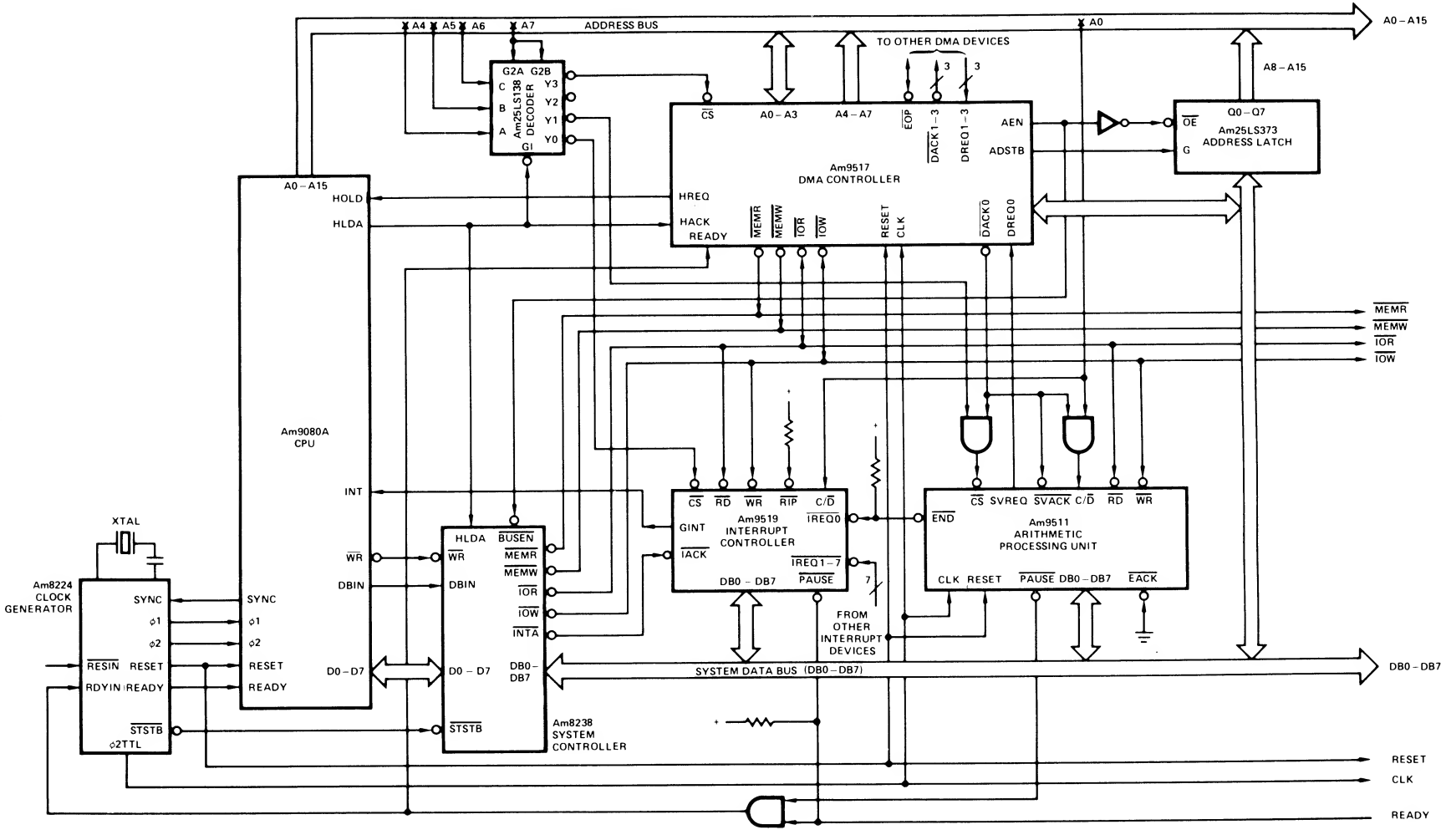


FIGURE 8.60 AMD 9511 interfacing: DMA configuration.

TABLE 8.11. AMD 9511 Commands

COMMAND SUMMARY

Command Code								Command Mnemonic	Command Description
7	6	5	4	3	2	1	0		
FIXED POINT 16 BIT									
sr	1	1	0	1	1	0	0	SADD	Add TOS to NOS. Result to NOS. Pop Stack.
sr	1	1	0	1	1	0	1	SSUB	Subtract TOS from NOS. Result to NOS. Pop Stack.
sr	1	1	0	1	1	1	0	SMUL	Multiply NOS by TOS. Lower half of result to NOS. Pop Stack.
sr	1	1	1	0	1	1	0	SMUU	Multiply NOS by TOS. Upper half of result to NOS. Pop Stack.
sr	1	1	0	1	1	1	1	SDIV	Divide NOS by TOS. Result to NOS. Pop Stack.
FIXED POINT 32 BIT									
sr	0	1	0	1	1	0	0	DADD	Add TOS to NOS. Result to NOS. Pop Stack.
sr	0	1	0	1	1	0	1	DSUB	Subtract TOS from NOS. Result to NOS. Pop Stack.
sr	0	1	0	1	1	1	0	DMUL	Multiply NOS by TOS. Lower half of result to NOS. Pop Stack.
sr	0	1	1	0	1	1	0	DMUU	Multiply NOS by TOS. Upper half of result to NOS. Pop Stack.
sr	0	1	0	1	1	1	1	DDIV	Divide NOS by TOS. Result to NOS. Pop Stack.
FLOATING POINT 32 BIT									
sr	0	0	1	0	0	0	0	FADD	Add TOS to NOS. Result to NOS. Pop Stack.
sr	0	0	1	0	0	0	1	FSUB	Subtract TOS from NOS. Result to NOS. Pop Stack.
sr	0	0	1	0	0	1	0	FMUL	Multiply NOS by TOS. Result to NOS. Pop Stack.
sr	0	0	1	0	0	1	1	FDIV	Divide NOS by TOS. Result to NOS. Pop Stack.
DERIVED FLOATING POINT FUNCTIONS									
sr	0	0	0	0	0	0	1	SQRT	Square Root of TOS. Result in TOS.
sr	0	0	0	0	0	1	0	SIN	Sine of TOS. Result in TOS.
sr	0	0	0	0	0	1	1	COS	Cosine of TOS. Result in TOS.
sr	0	0	0	0	1	0	0	TAN	Tangent of TOS. Result in TOS.
sr	0	0	0	0	1	0	1	ASIN	Inverse Sine of TOS. Result in TOS.
sr	0	0	0	0	1	1	0	ACOS	Inverse Cosine of TOS. Result in TOS.
sr	0	0	0	0	1	1	1	ATAN	Inverse Tangent of TOS. Result in TOS.
sr	0	0	0	1	0	0	0	LOG	Common Logarithm (base 10) of TOS. Result in TOS.
sr	0	0	0	1	0	0	1	LN	Natural Logarithm (base e) of TOS. Result in TOS.
sr	0	0	0	1	0	1	0	EXP	Exponential (e^X) of TOS. Result in TOS.
sr	0	0	0	1	0	1	1	PWR	NOS raised to the power in TOS. Result in NOS. Pop Stack.
DATA MANIPULATION COMMANDS									
sr	0	0	0	0	0	0	0	NOP	No Operation
sr	0	0	1	1	1	1	1	FIXS	Convert TOS from floating point to 16-bit fixed point format.
sr	0	0	1	1	1	1	0	FIXD	Convert TOS from floating point to 32-bit fixed point format.
sr	0	0	1	1	1	0	1	FLTS	Convert TOS from 16-bit fixed point to floating point format.
sr	0	0	1	1	1	0	0	FLTD	Convert TOS from 32-bit fixed point to floating point format.
sr	1	1	1	0	1	0	0	CHSS	Change sign of 16-bit fixed point operand on TOS.
sr	0	1	1	0	1	0	0	CHSD	Change sign of 32-bit fixed point operand on TOS.
sr	0	0	1	0	1	0	1	CHSF	Change sign of floating point operand on TOS.
sr	1	1	1	0	1	1	1	PTOS	Push 16-bit fixed point operand on TOS to NOS (Copy)
sr	0	1	1	0	1	1	1	PTOD	Push 32-bit fixed point operand on TOS to NOS. (Copy)
sr	0	0	1	0	1	1	1	PTOF	Push floating point operand on TOS to NOS. (Copy)
sr	1	1	1	1	0	0	0	POPS	Pop 16-bit fixed point operand from TOS. NOS becomes TOS.
sr	0	1	1	1	0	0	0	POPD	Pop 32-bit fixed point operand from TOS. NOS becomes TOS.
sr	0	0	1	1	0	0	0	POPF	Pop floating point operand from TOS. NOS becomes TOS.
sr	1	1	1	1	0	0	1	XCHS	Exchange 16-bit fixed point operands TOS and NOS.
sr	0	1	1	1	0	0	1	XCHD	Exchange 32-bit fixed point operands TOS and NOS.
sr	0	0	1	1	0	0	1	XCHF	Exchange floating point operands TOS and NOS.
sr	0	0	1	1	0	1	0	PUPI	Push floating point constant "π" onto TOS. Previous TOS becomes NOS.

NOTES:

1. TOS means Top of Stack. NOS means Next on Stack.
2. AMD Application Brief "Algorithm Details for the Am9511 APU" provides detailed descriptions of each command function, including data ranges, accuracies, stack configurations, etc.
3. Many commands destroy one stack location (bottom of stack) during development of the result. The derived functions may destroy several stack locations. See Application Brief for details.
4. The trigonometric functions handle angles in radians, not degrees.
5. No remainder is available for the fixed-point divide functions.
6. Results will be undefined for any combination of command coding bits not specified in this table.

9

analog interfacing

Many important applications of microcomputers, particularly in industrial process control or laboratory applications, require that the computer be interfaced with an input or output device that operates in a analog manner. It would be worthwhile to put the issue of analog interfacing in perspective by first considering a highly simplified block diagram of such a system, shown in Figure 9.1, before turning a more detailed description of the structure and operation of analog interfaces.

"The basic input of such a system is the sensor element or transducer, which measures some continuously variable physical parameter such as temperature, pressure, velocity, and radiation flux. The transducer functions to convert the magnitude of the physical variable into a continuously variable electrical parameter such as voltage or current.

To be handled by the microcomputer system, the continuously varying voltage must be converted into a digit signal. The conversion is performed by a unit known as an analog to digital (A/D) converter. A/D converters are typically implemented on printed circuit cards that fit into the mainframe of the microcomputer system. Such boards include multiplexers, so that a plurality of discrete analog inputs or "channels" can be handled.

"After the digital signals are processed by the microsary to display, record, or make use of the data. Such devices include displays, chart recorders, and industrial controls that operate in an analog manner. To interface to such devices, a digital to analog (D/A) converter is used. D/A converters are likewise implemented on printed circuit cards, and in some cases a single card may serve the purpose of both an A/D and a D/A converter. The analog output of such cards is connected to an actuator, as shown in Figure 9.1, which is used to perform some industrial control functions."

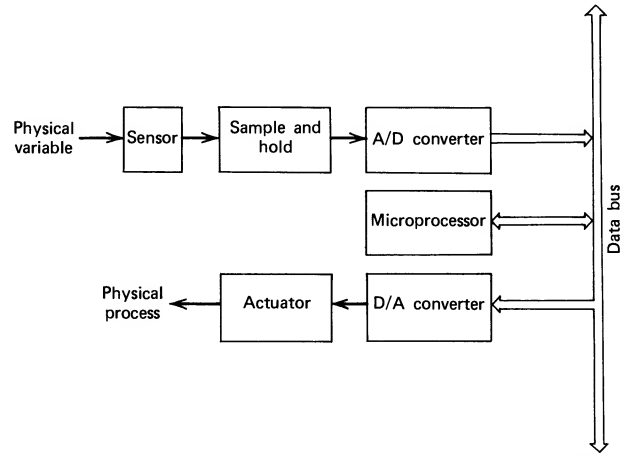


FIGURE 9.1 Microcomputer interfacing to an analog system.

In this chapter we review some of the important features of an analog interfacing system, in particular:

- Sensors.
- Data conversion devices.
- Analog signal processing.

"Sensors

Sensors are the means for measuring some physical variable such as temperature, pressure, position, fluid flow, and environmental conditions. There are a number of different physical technologies for making such measurements, which differ in accuracy, sensitivity, and cost. These technologies are summarized, together with their advantages and disadvantages, in Table 9.1."

Table 9.1. A Characterization of Sensors[a]

Device	*Principle of Operation*	*Advantages*	*Disadvantages*
Temperature			
Thermocouple	EMF generated at the junction of two dissimilar materials by the Thompson effect	Largest temperature range, very high temperatures	Small signal, low sensitivity, lowest accuracy, unstable, expensive
Thermistor	Change in resistance with temperature in semiconducting metal oxides	Largest signal, high sensitivity, low cost, modest accuracy	Nonlinear, limited temperature range, part-to-part variation
Resistance temperature detector (RTD)	Change in resistance with temperature in metallic conductors	Wide temperature range, highest accuracy, very stable, linear	Modest signal, high cost, long thermal time constant
Semiconductor	Change in V_{be} with temperature in forward-biased diode	Good accuracy, highly stable, modest cost, manufactured using standard semiconductor processing methods	Small signal, part-to-part variation, nonlinear, requires external signal conditioning
Thyristor temperature switch	Thermal breakdown of thyristor gate at set temperature and voltage	Capable of handling large currents	Set point depends on doping profile of junction, limited temperature range
Spreading resistance sensor (SRS)	Change in resistance of extrinsic silicon with temperature	Manufactured using common planar processing methods, fast thermal response, comparable to RTDs in performance, potentially low cost	Requires highly uniform substrate material, high-resolution photolithography
Pressure			
Potentiometric	Change in resistance of slidewire resistor coupled to diaphragm structure of Bourdon tube	Low cost, relatively insensitive to temperature, large output signal	Low accuracy, low sensitivity, hysteresis, noise, wear
Reluctive (LVDT)	Change in reluctance due to motion of core slug coupled to diaphragm structure or Bourdon tube	Modest cost, temperature insensitive, large output ac signal	Sensitive to shock and vibration, ac output, slow response time
Bonded strain gauge	Change in resistance due to pressure-differential-induced strain in diaphragm structure	High sensitivity, low hysteresis, can have differential output, ratiometric to supply voltagw	Small output signal, temperature sensitive, can be unstable due to bond degradation, costly
Capacitive	Change in capacitance due to deflection of diaphragm constituting one plate of a parallel-plate capacitor	Highest sensitivity, can be used in harsh environments, temperature insensitive	Typically very low capacitance, difficult to manufacture, expensive in most applications
Piezoresistive strain gauge	Change in resistance due to pressure-induced strain in integral diaphragm structure	Potentially low cost, high sensitivity, low hysteresis, avoids problems of bonded strain gauge, can be manufactured using planar processing technology	Very temperature sensitive, low-output signal, difficult to package, possible environmental problems

(*Continued*)

Table 9.1 (*Continued*)

Device	*Principle of Operation*	*Advantages*	*Disadvantages*
Position			
Potentiometric	Change in resistance of slide-wire resistor coupled to mechanical motion	Low cost, relatively temperature insensitive, large output signal	Low accuracy, low sensitivity, hysteresis, noise, wear
Reluctive	Change in Q of coil due to motion of core slug (LVDT) or proximity of metallic material	Modest cost, temperature insensitive, large output signal, good accuracy	Sensitive to shock and vibration (LVDT), requires metallic material, ac output
Optical	Interruption or reflection of optical signal to optoelectronic detector	Noncontacting, can sense nonmetallic materials, fast response time	Subject to contamination by dirt, oil, and other opaque materials
Hall effect	Hall effect in semiconducting element generates output voltage proportional to field strength	Noncontacting, not subject to interference by dirt and oil, can be manufactured in integrated form using semiconductor methods	Requires external magnet, temperature sensitive, small output signal, difficult to package
Wiegand effect	Domain reversal in specially processed ferrous materials, generates large voltage pulse	Noncontracting, temperature insensitive, requires no external power supply, large output signal	Pulse output only, poorly understood phenomenon, requires external magnet
Magnetodiode	Change in V_{be} or I_r of p-i-n or Schotky diode due to minority carrier recombination at magnetic-barrier layer structure	Noncontacting, large output signal, can be incorporated in discrete devices such as UJTs for switching	Poorly understood phenomenon, requires external magnet
Fluid flow			
Turbine or vane	Momentum transfer from fluid flow to rotating turbine or movable vane	Mass flow, linear output can provide digital signal (pulse train)	Requires calibration for specific fluid, can restrict flow
Vortex shedding	Bluff body induces turbulent vortices in flow that can be sensed as pressure or velocity changes	Simple, no moving parts, digital output (pulse train)	Provides velocity flow only
Thermal transfer	Heat transfer by fluid from heating element, can be sensed as change in resistance of heater element or by separate temperature sensor	Simple mass-flow sensor, no moving parts	Highly nonlinear, requires calibration for specific fluid
Differential pressure	Pressure differential across orifice in flow stream	Uses conventional pressure sensors in differential mode	Provides velocity flow only, highly nonlinear
Environmental			
Semiconducting metal-oxide gas	Change in conductivity due to surface adsorption of gaseous elements	Simple solid-state device, simple signal conditioning, low cost	Only partially selective, affected by several gaseous elements, part-to-part variation
Alumina humidity	Change in impedance of anodized alumina thin film due to adsorption of water vapor	Simple solid-state device, can be incorporated in conventional semiconductor devices	Unstable, part-to-part variation, high-temperature limitations
Brady array humidity	Change in conductivity of semiconducting oxide element due to adsorption of water vapor	High sensitivity, fast response	Expensive, part-to-part variation, complex processing of signal
MOSFET gas	Change in flat-band voltage of MOSFET due to surface adsorption of gaseous elements	Standard semiconductor device highly sensitive, easily measured, can be incorporated in standard IC products	Sensitive to many different chemical elements, phenomenon poorly understood
Surface junction	Change in semiconductor junction properties due to adsorption at surface junction	Standard semiconductor device, highly sensitive, easily measured, can be incorporated in standard IC products	Sensitive to many different chemical elements, effect difficult to control

[a]*SOURCE: IEEE Spectrum, November 1977, pp. 42–43. © IEEE 1977. Reprinted with permission.*

Data Conversion Devices

"A/D Converters

There are a number of different methods of A/D conversion: counter or servo, successive approximation, dual slope/quad slope, parallel voltage-to-frequency conversion, and charge balancing.

"The counter or servo type of A/D converter utilizes a counter that begins a count at a predetermined time, is converted into an analog signal, and is compared with the input analog signal. When the output is equal to the input, the comparator changes state, and the state of the counter represents the output digital number. The disadvantage of such a converter is its relatively low speed.

"The successive-approximation type operates by successively comparing the input voltage with a reference signal 1 bit at a time. Successive-approximation A/D converters have good resolution and high speed.

"The dual slope and quad slope A/D converters are examples of the integrating A/D converter which operates by converting a voltage to a time period using an integrator. These converters are low cost and are frequently found in digital voltmeters or digital panel meters. The main disadvantage of the integrating-type converter is its relatively long conversion time.

"The parallel A/D converter utilizes a plurality of comparators in parallel to provide the fastest possible conversion.

"The voltage-to-frequency and charge-balancing types of A/D converters are related methods that involve charging a capacitor with a current proportional to the input voltage and then determining its charge."

The successive approximation A/D converter operates by making a comparison between the analog input signal and a feedback signal provided by the circuit. A sequence of comparisons are made, with each comparison making closer and closer approximations to the actual digital value of the analog input signal, and hence the name "successive approximation."

Although there are a number of different implementations, the basic structure of the successive approximation A/D converter consists of a shift register. The output of the shift register is applied to a storage register that provides a digital signal to a digital-to-analog converter. An example of a digital-to-analog converter used in such a circuit is a resistor ladder network. The resistor ladder network produces an analog signal corresponding in magnitude to the value of the digital signal stored in the storage register. This analog signal is then applied to a comparator where comparison is made between the unknown analog input signal and the known analog signal derived from the digital-to-analog converter.

The comparator determines whether the supplied analog reference signal is greater or less than the analog input signal. The results of the comparison are applied to a control logic circuit, which stops the shift register and allows the digital signal value corresponding to the analog input signal to be output, as shown in Figure 9.2.

In comparing the performance of successive approximation A/D converters, various technical parameters may be used for developing an overall figure of merit.

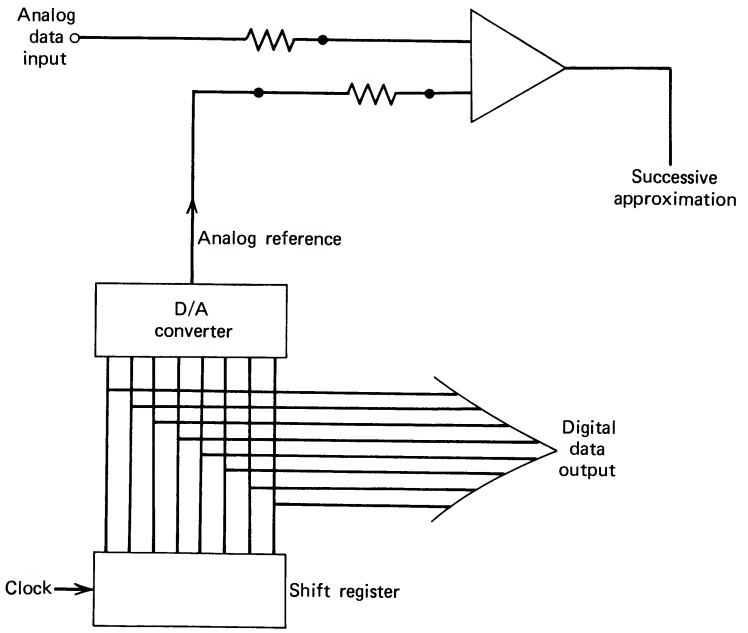


FIGURE 9.2 Successive approximation A/D converter.

One useful figure of merit is the accuracy-speed product. The speed-accuracy product is principally effected by the comparator. The principal limitation on the amplifier is the time in which the comparator has to switch from 0 V to the threshold voltage. The comparator may be either overdamped or underdamped. For fastest operation, however, the amplifier is critically damped or slightly underdamped. It must be realized that the key operational factor in the operation of the comparator is to change the feedback voltage from a given value to its original level. This frequently requires slew rates greater than 10 V/μsec.

It is clear, therefore, that there is a trade-off between conversion speed and accuracy. How does this trade-off change as the parameters are changed? Essentially, the accuracy is principally determined by the feedback loop, and in particular the resistor network. At low conversion rates various static parameters of the ratio tolerances of the resistor ladder and mismatch of input and output impedances that generate the errors. At higher conversion rates, however, other factors are involved. These include actual delays in the resistor ladder network and transients at the switches. The higher the conversion rate, the more significant such transients become so that an A/D converter operating at the rate of 10,000 conversions/sec may only achieve an accuracy of 0.05% over the full rated range.

It is important to make reference to the efficiency of the successive approximation A/D converter. It must be first pointed out that basically a conversion cycle requires N clock periods, and the conversion time is equal to N/F, where N is the number of bits and F is the clock frequency. It is therefore apparent that the accuracy of the conversion, and the time required to perform the conversion, are inversely proportional. The accuracy of the conversion process is a function of several parameters in addition to the number of bits. It is basically a function of the quantizing error, and errors arising from inaccuracies in the D/A decoder, the analog comparator, or even the reference voltage. The typical accuracies with current commercially available successive approximation A/D converters are about 0.005%.

The dual slope device operates in three phases. During the first phase the converter is reset so that the analog components are automatically nulled out. The input is grounded and a feedback loop results in any error information being stored by a capacitor. In the second phase the input signal is applied to the converter and is integrated over a predetermined number of pulses producing an output voltage directly proportional to the input voltage. In the third phase the input of the converter is switched from the applied input signal voltage to a reference voltage. The output of the integrator then discharges back to zero because of the polarity of the reference voltage. Included in the circuit is a counter that counts the number of clock pulses between the beginning of the third phase and the time when the integrator output passes through zero. The number of digital clock pulses is a digital measure of the magnitude of the input voltage. The basic block diagram of the integrating converter is shown in Figure 9.3. A comparison of the conversion times for the two types of converters as a function of their accuracy, for the number of bits in the converter, is shown in Table 9.2.

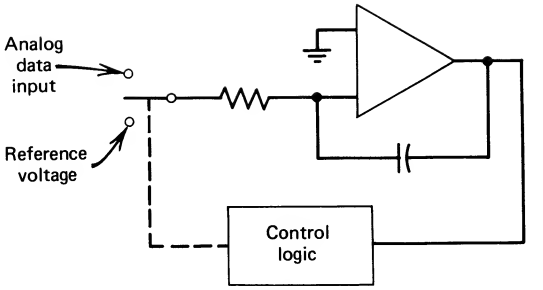


FIGURE 9.3 Integrating A/D converter.

Table 9.2. A/D Converters

Type of Converter	*Speed Category*	*Conversion Time* 8 Bits	10 Bits (msec)	12 Bits (msec)	16 Bits (msec)
Integrating	Slow	20 msec	30	30	250
	Medium	1 msec	5	20	–
	Fast	0.3 msec	1	5	–
Successive approximation	General purpose	30 μsec	40	50	–
	High performance	10 μsec	15	20	400
	Fast	5 μsec	10	12	–
	High speed	2 μsec	4	6	–
	Ultra fast	0.8 μsec	1	2	–

"D/A Converters

D/A converters can be divided into two categories: current output and voltage output types. The actual technique of D/A conversion is somewhat simpler than A/D conversion. The digital signal is applied to a precision ladder network of resistors, to which a precise reference voltage source is applied. The output of the network is the sum of all the binary wights in the form of a voltage or current."

Implementations

There are four distinct types of data conversion implementations:

- Monolithic chips.
- Hybrid circuits.
- Modular devices.
- Boards.

The monolithic devices are usually low-cost general-purpose units and are usually directed to 8-bit applications. Generally, the single chip devices are successive approximation type A/D converters. Such converters usually need one or more external components such as a voltage reference, a comparator, and a clock. The accuracy ranges from 8 to 12 bits and the converters operate at a conversion time of between 20 and 40 μsec.

Another type of monolithic single chip converter is the integrating A/D converter. Integrating converters are somewhat slower than successive approximation converters, but they have very good accuracy. The integrating converter operates by producing an output signal that is the integral of the input voltage over a predetermined period of time. Although there are numerous types of integrating converters the most widely used is the dual slope converter.

The monolithic digital-to-analog converters are generally implemented using resistor networks. The more sophisticated include a voltage reference and an output amplifier. The output is generally in the form of a current.

Next we consider the hybrid data converter. In general hybrids have higher performance but greater complexity. A monolithic converter has a settling time as fast as 0.5 μsec for an 8-bit converter or 100 μsec for a 16-bit converter. The hybrid A/D converters, generally successive approximation devices, offer a conversion time from under 1 μsec to about 50 μsec.

The modular components generally offer the greatest performance, yet are correspondingly expensive and fairly large. For the digital-to-analog converters settling times range from 2 to 750 nsec for current output units and 5 to 250 μsec for voltage output types. The modular analog-to-digital converters come in a number of different circuits, including successive approximations, integrating, and high speed parallel conversion. The conversion times range from 40 to 100 nsec for parallel types at 4–8 bits of accuracy to 300 μsec to 250 msec for integrating types with accuracies of 8–14 bits.

The final type of implementation to consider is the analog board. Such boards are compatible with single board I/O systems and provide multichannel analog-to-digital and digital-to-analog capability. Such boards incorporate a complete digital interface and therefore are the simplest way to implement a data conversion function.

Analog Signal Processing

There is increasing interest in the replacement of active and passive analog filters, such as might be found in communications systems, with digital filters, thereby reducing the size, cost, and power required in such applications. One of the new circuits that make such processing possible is the Intel 2920 real time signal processor.

Intel 2920

A block diagram of the Intel 2920 real time signal processor is shown in Figure 9.4. The device was announced at the 1979 International Solid State Circuits Conference (ISSCC).

The device basically combines an analog-to-digital and a digital-to-analog converter with a digital processor, and all are implemented on a single monolithic semiconductor chip. An on-board electrically programmable read-only memory is provided that contains a user program that performs the specific digital signal processing required by the application.

The device provides four analog inputs that are converted into digital signals by means of a successive approximation digital-to-analog converter. The processor itself is a 28-bit arithmetic logic unit that digitally processes data up to 25-bit precision. A scratch pad random access memory containing forty 25-bit words is also provided.

The instructions are 24 bits in length and are formatted into five contiguous fields:

- The digital operator.
- The source address.
- The destination address.
- The amount of shifting.
- The analog operator.

The on-board E PROM can store up to 192 such instructions. Since each instruction executes in 400 nsecs at the highest permissible clock speed, the longest program (192 instructions in length) executes in 76.8 μsecs. This parameter is the basic parameter limiting the accuracy of the system according to Nyquist's theorem. This theorem states that a continuous system may be simulated by sampling only if the sampling rate is greater than twice the highest frequency of the signals being

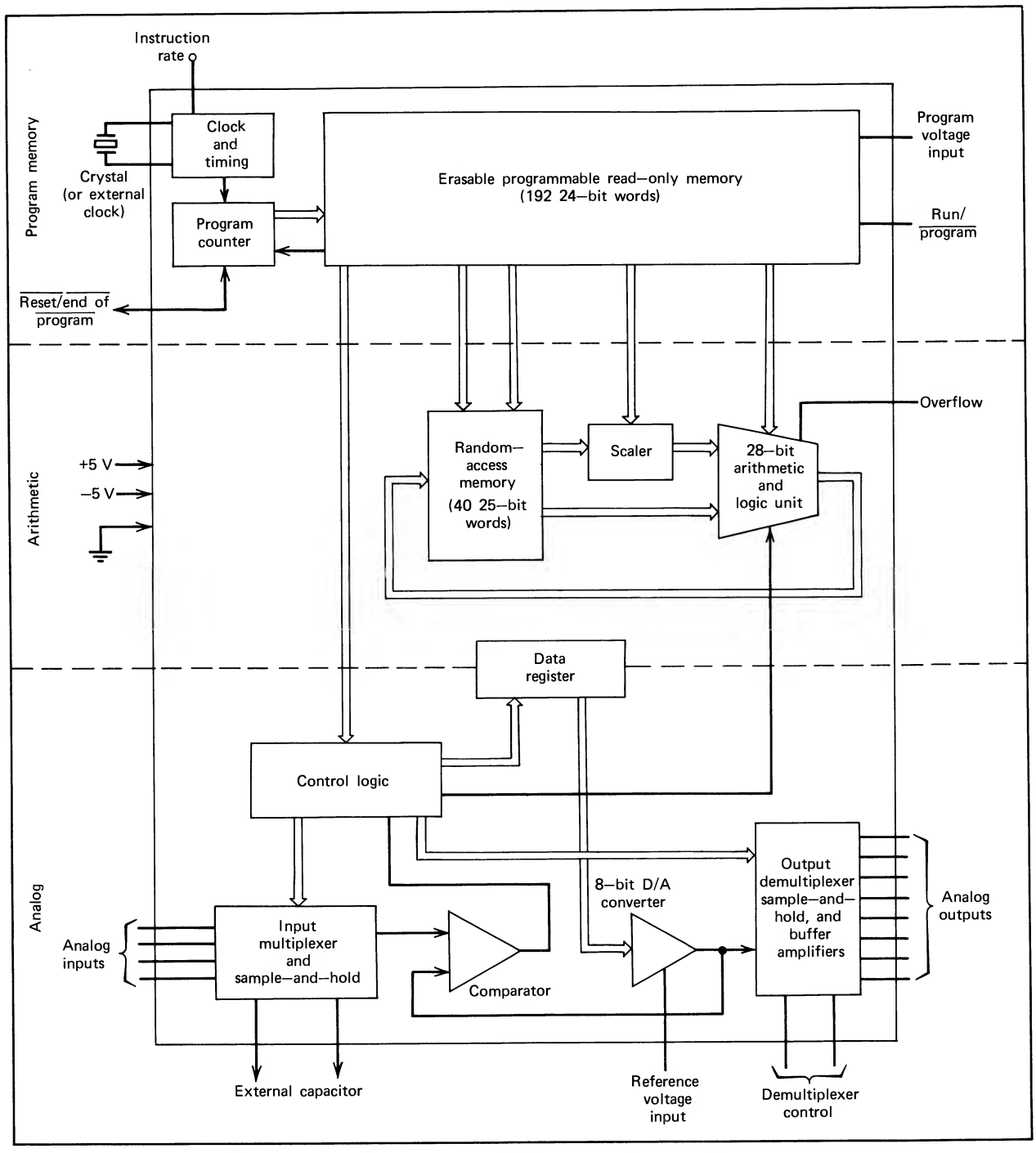


FIGURE 9.4 Intel 2920 functional block diagram. (Courtesy Intel Corporation.)

processed. The worst case bandwidth of the chip is therefore 1/76.8 μsecs, or 6.5 KHz. The minimum sampling rate that must be performed to achieve an accurate simulation is therefore 13 KHz. The results of the operation are some modification of the digital signal, which is then provided to a digital-to-analog converter and in turn to an output demultiplexer, sample and hold, and buffer amplifier circuit. The demultiplexer provides eight buffered outputs.

10

bus standards

Multibus

"The Intel Multibus was introduced with the SBC 80/10 single-board computer and is primarily of interest in the industrial microcomputer market.

"The layout of the Multibus card is shown in Figure 10.1 and the pin definitions are given in Table 10.1. It is noted that an additional 60-pin edge connector is provided as user-definable pins. An example of a multibus printed circuit card is the Intel SBC 064 RAM module, shown in Figure 10.2, for use with the Intel single-board computers.

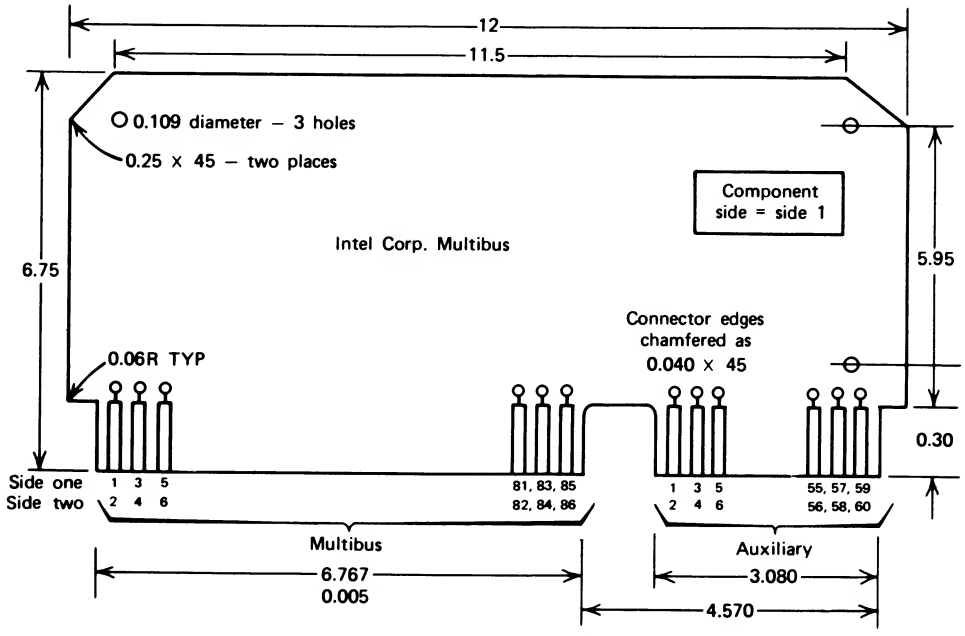


FIGURE 10.1 Intel Multibus circuit card.

Table 10.1. Intel Multibus Pin Definitions

	Edge Connector Pin List					
	Component Side			Circuit Side		
	Pin	Mnemonic	Description	Pin	Mnemonic	Description
	1	GND	Signal GND	2	GND	Signal GND
	3	+5	+5 V dc	4	+5	+5 V dc
Power	5	+5	+5 V dc	6	+5	+5 V dc
supplies	7	+12	+12 V dc	8	+12	+12 V dc
	9	5	5 V dc	10	5	5 V dc
	11	GND	Signal GND	12	GND	Signal GND
	13	BCLK/	Bus clock	14	INIT/	Initialize
	15	BPRN/	Bus priority in	16	BPRO/	Bus priority out
	17	BUSY/	Bus busy	18	BREQ/	Bus request
	19	MRDC/	Memory read command	20	MWTC/	Memory write command
Bus	21	IORC/	Input/Output read command	22	IOWC/	Input/Output write command
controls	23	XACK/	Xfer acknowledge	24	INH1/	Inhibit 1 disable RAM
	25	AACK/	Special acknowledge	26	INH2/	Inhibit 2 disable PROM or ROM
	27		Reserved	28		Reserved
	29		Reserved	30		Reserved
	31	CCLK/	Constant clock	32		Reserved
	33		Reserved	34		Reserved
	35	INT6/		36	INT7/	
	37	INT4/		38	INT5/	
Interrupts	39	INT2/	Parallel interrupt requests	40	INT3/	Parallel interrupt requests
	41	INT0/		42	INT1/	
	43	ADRE/		44	ADRF/	
	45	ADRC/		46	ADRD/	
	47	ADRA/	Address bus	48	ADRB/	Address bus
Address	49	ADR8/		50	ADR9/	
	51	ADR6/		52	ADR7/	
	53	ADR4/		54	ADR5/	
	55	ADR2/		56	ADR3/	
	57	ADR0/		58	ADR1/	
	59	DATE/		60	DATF/	
	61	DATC/		62	DATD/	
	63	DATA/	Data bus	64	DATB/	Data bus
	65	DAT8/		66	DAT9/	
Data	67	DAT6/		68	DAT7/	
	69	DAT4/		70	DAT5/	
	71	DAT2/		72	DAT3/	
	73	DAT0/		74	DAT1/	
	75	GND	Signal GND	76	GND	Signal GND
	77	10*	10 V dc	78	10*	10 V dc
Power	79	12	12 V dc	80	12	12 V dc
supplies	81	+5	+5 V dc	82	+5	+5 V dc
	83	+5	+5 V dc	84	+5	+5 V dc
	85	GND	Signal GND	86	GND	Signal GND

SOURCE: Intel Corporation.

Some of the other notable features are: well-laid-out address and data buses, and a substantial number of power supply and ground pins. As for the LSI-11, there are also a number of second sources for memory and peripheral boards, as well as the processor board (single-board computer) itself.

"The drawbacks to the Multibus are also similar: it is specifically configured for the 8080/8085 microprocessor system, and the cost of the boards is relatively higher than that of other configurations."

"LSI-11 Bus

"Another important bus configuration is the Digital Equipment Corporation LSI-11, which is used in the

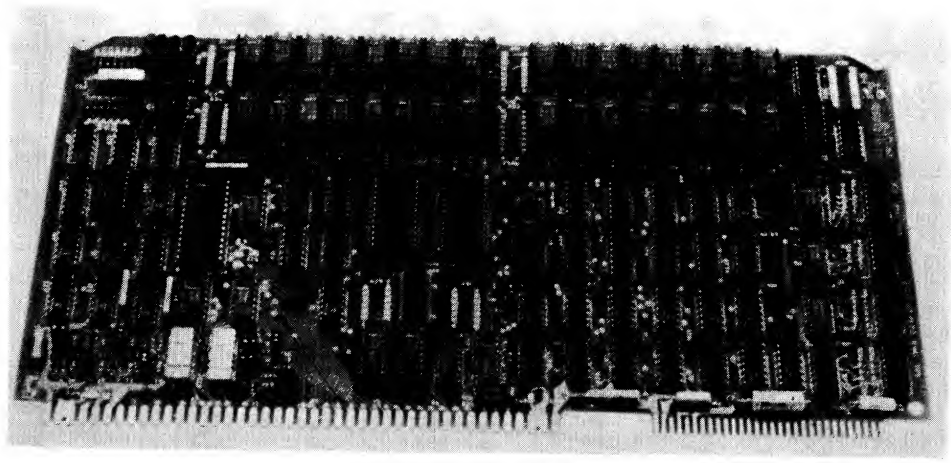


FIGURE 10.2 Intel SBC 064 RAM card. (Courtesy Intel Corporation.)

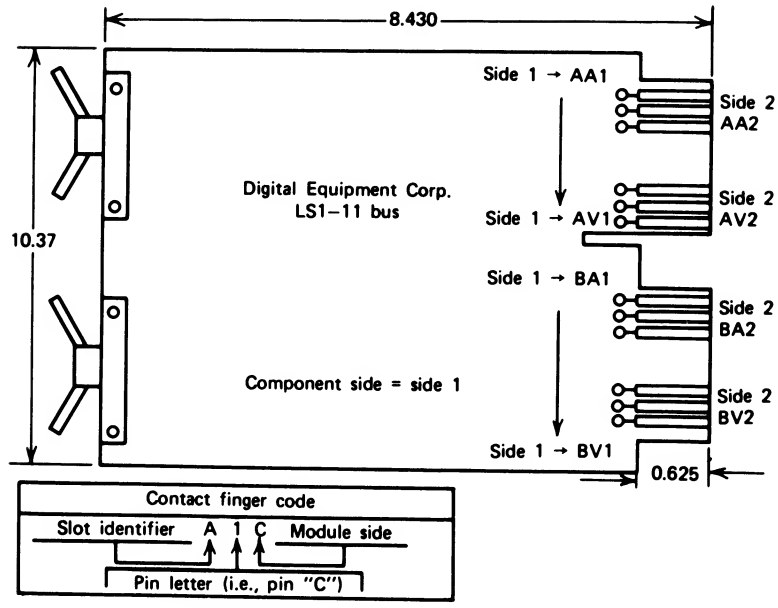


FIGURE 10.3 DEC LSI-11 circuit card.

LSI-11 microcomputer. The LSI-11 is of interest to personal computer users since it is used in the Heath H-11 system, as well as being an industrial and OEM microcomputer.

"The layout of the LSI-11 card is shown in Figure 10.3 and Table 10.2. Note that the card uses a "gapped" pin configuration. The standard card consists of 72 pins, as shown in Figure 10.3, but a half-card (36 pins) and a double card (144 pins) configuration can also be used.

"Some of the notable features of the LSI-11 bus are: a multiplexed address and data bus, a substantial number of power supply and ground pins, and a number of user-definable pins. Moreover, the LSI-11 configuration is produced by a number of second sources for memory and peripheral boards.

"There are some drawbacks to the LSI configuration: it is only intended for the LSI-11 computer, and the costs of the memory and peripheral boards are relatively higher than those of other configurations."

Table 10.2. LSI-11 Pin Definitions

Edge Connector Pin List		
Line No.	*Pin No.*	*Signal Description*
1	AA1	BSPARE 1—unassigned line (bus)
2	AB1	BSPARE 2—unassigned line (bus)
3	AC1	BAD16— } extended address bits
4	AD1	BAD17— } (not implemented)
5	AE1	SSPARE 1 } special unbused spares for user
6	AF1	SSPARE 2 } interconnections
7	AH1	SSPARE 3 }
8	AJ	GND—system common (signal, dc, etc.) ground
9	AK1	MSPARE A } maintenance spares normally
10	AL1	MSPARE A } connected together
11	AM1	GND—system common (signal, dc, etc.) ground
12	AN1	BDMRL—direct memory access request
13	AP1	BHALTL—processor halt enable
14	AR1	BREFL—memory refresh line
15	AS1	+12B—+12 V dc battery power
16	AT1	GND—system common (signal, dc, etc.) ground
17	AU1	PSPARE 1—spare but use not recommended
18	AV1	+5B—+5 V dc battery power
19	AA2	+5—normal +5 V dc system power
20	AB2	-12—normal -12 V dc system power
21	AC2	GND—system common (signal, dc, etc.) ground
22	AD2	+12—normal +12 V dc system power
23	AE2	BDOUT L—implies valid data on BDALO
24	AF2	BRPLY L—BDOUT/BDIN reply flag
25	AH2	BDIN L—indicates input transfer or interrupt operation
26	AJ2	BSYNC L—Bus master/address synchronizer
27	AK2	BWT BTL—write byte indicating data flow or addressing
28	AL2	BIRQ L—interrupt request
29	AM2	BIAKI L— } input and output interrupt acknow-
30	AN2	BIAKO L— } ledge
31	AP2	BBS7 L—bank 7 select
32	AR2	BDMGI L— } data grant input and output
33	AS2	BDMGOL— }
34	AT2	BINIT L—initializes or clears all input/output buses on power up
35	AU2	BDAL0 L— } data/address lines
36	AV2	BDAL1 L— }

Table 10.2 (*Continued*)

Edge Connector Pin List		
Line No.	*Pin No.*	*Signal Description*
37	BA1	BDCOK H—dc voltage OK flag
38	BB1	BPOK H—power OK flag
39	BC1	BSPARE 4— } special unbused spare lines for users
40	BD1	BSPARE 5— }
41	BE1	BSPARE 6— }
42	BF1	BSPARE 7— }
43	BH1	BSPARE 8— }
44	BJ1	GND—system common (signal, dc, etc.) ground
45	BK1	MSPAREB— } Maintenance spares connected
46	BL1	MSPAREB— } together
47	BM1	GND—system common (signal, dc, etc.) ground
48	BN1	BSACK L—indicates DMA device is bus master
49	BP1	BSPARE 6—bus spare not for use
50	BR1	BEVNT L—external event interrupt request
51	BS1	PSPARE 4—spare, use not recommended
52	BT1	GND—system common (signal, dc, etc.) ground
53	BU1	PSPARE 2—spare, use not recommended
54	BV1	+5—normal +5 V dc system power
55	BA2	+5—normal +5 V dc system power
56	BB2	-12—normal -12 V dc system power
57	BC2	GND—system common (signal, dc, etc.) ground
58	BD2	+12—normal +12 V dc system power
59	BE2	BDAL2— } data/address lines
60	BF2	BDAL3— }
61	BH2	BDAL4— }
62	BJ2	BDAL5— }
63	BK2	BDAL6— }
64	BL2	BDAL7— }
65	BM2	BDAL8— }
66	BN2	BDAL9— }
67	BP2	BDAL10— }
68	BR2	BDAL11— }
69	BS2	BDAL12— }
70	BT2	BDAL13— }
71	BU2	BDAL14— }
72	BV2	BDAL15— }

SOURCE: DEC.

11

system development

System development is the term used for the design and development of a microcomputer system starting from a microprocessor and other integrated circuits. The system designer can approach the task from various starting points:

- Components (microprocessor, memory, I/O devices)
- Parts families (sets of related components provided by the manufacturer).
- Boards (printed circuit boards with general-purpose capabilities).

Regardless of from which level the designer starts, the process of system development is the same: determining which microprocessor and support components are most suitable for the intended application; interfacing the selected components to the application; and configuring and optimizing the resulting system. The present chapter is concerned with the equipment for performing these tasks.

In the present analysis of system development, we consider five distinct types of equipment:

- Evaluation systems.
- Microprocessor development systems.
- Single board computers.
- Logic analyzers.
- Field test equipment.

Evaluation Systems

An evaluation system, as the name implies, is a inexpensive device that enables the user to become familiar with a particular microprocessor and "evaluate" it for use in a given application. An evaluation system typically consists of the microprocessor, some RAM, a ROM containing a control program, and input/output lines, implemented on a single PC board. Since evaluation sys-

tems are intended to be programmed directly in machine language, they often include a simple calculator-type keypad and LED display.

MCS-86 System Design Kit (SDK-86)

An example of an evaluation system for the Intel 8086 microprocessor is the MCS-86 System Design Kit (SDK-86). The SDK-86 has the following features:

CPU: **8086**
Memory
8K bytes ROM
2K bytes RAM
Memory addressing
ROM: FE000 through FFFFF
RAM: 0 through 7FF
Input/output
Parallel: 48 lines
Serial: RS232 or current loop
User interface
24-key keypad
8-digit LED display

A photograph of the actual SDK-86 unit appears in Figure 11.1.

Because of the importance of evaluation systems in system development, it is worthwhile to describe the structure and operation of the SDK-86 in greater detail. A block diagram of the SDK-86 is shown in Figure 11.2, to which we now turn our attention.

In the upper left hand corner of Figure 11.2 is the 8284 clock generator. As shown in the diagram, the 8284 operates together with a 14.7456 quartz crystal to produce a 4.9-MHz CLK signal that is used by the

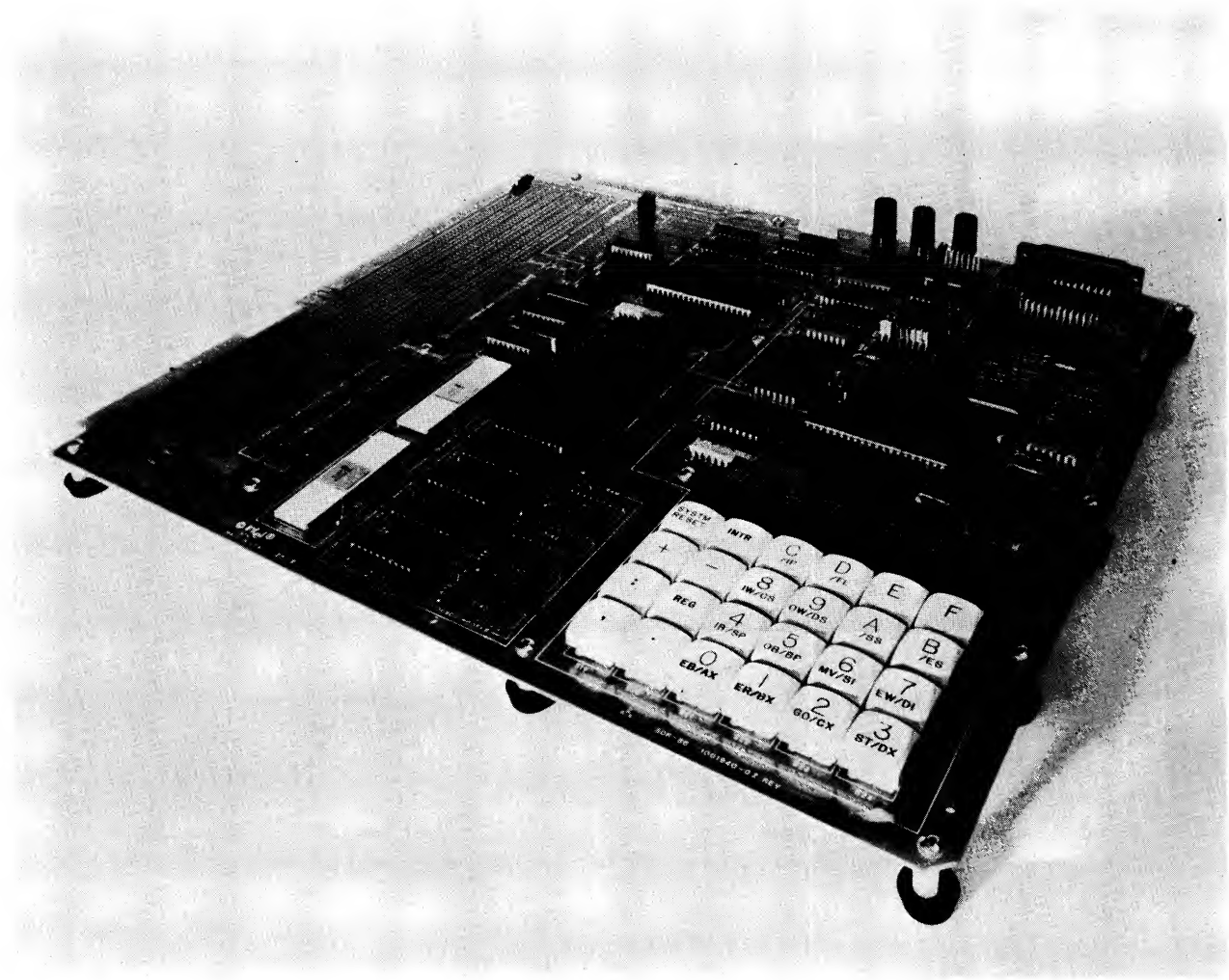


FIGURE 11.1 Intel SDK-86. (Courtesy Intel Corporation.)

8086 CPU. A jumper link labeled W40/W41, is provided so that the user may choose either low-speed (2.45 MHz) or regular (4.9 MHz) operation.

The clock generator is connected to a 74LS164 wait state generator as well as the 8086 microprocessor. The function of the wait state generator is to insert wait states into the bus cycle of the CPU to interface with a slower peripheral or memory circuit. A shorting plug is used to select the appropriate number of wait states for suitable operation.

Since the details of the 8086 microprocessor are discussed in preceding chapters, it is not necessary to repeat such information here. It is, however, instructive to note the particular implementation here. The MN/$\overline{MX}$ pin has been tied to V_{cc} (logical 1), and therefore the 8086 is in minimum mode. The nonmaskable interrupt (NMI) input to the 8086 originates from the INTR or interrupt key on the keypad.

Extending in the horizontal direction are the control lines and the address and data bus. Parallel I/O ports are formed using the programmable peripheral interface circuits described above. Each of the integrated circuits contains three data ports that can be individually addressed. The ports associated with the first integrated circuit are P1A, P1B, and P1C. The ports associated with the second integrated circuit are P2A, P2B, and P2C. In addition, each of the integrated circuits contains a write-only control port. This control port is labeled "P1 control" and "P2 control" in the first and the second integrated circuit respectively. Table 11.1 shows the individual I/O port addresses corresponding to the designations used above.

Table 11.1. SDK-86 Parallel I/O Port Addresses

Port	*Address*	*Port*	*Address*
P2A	FFF8	P1A	FFF9
P2B	FFFA	P1B	FFFB
P2C	FFFC	P1C	FFFD
P2 control	FFFE	P1 control	FFFFF

In the upper right-hand portion of the figure is a representation of the random access memory. In the present implementation such RAM is specified as being at the low end of memory, that is, storing location 0 through 07FF. Such RAM circuits are enabled by an appropriate output from the RAM decode circuit.

Another memory in the system is the read-only memory shown as the 2316/2716 circuits located in the center of the figure. Such circuits contain the keypad monitor program and the serial monitor program. The ROM is enabled by a predetermined output from the decode circuit attached thereto. The monitor memory allocation in the ROM is shown in Figure 11.3.

The other circuits shown in the figure include an I/O decode circuit for generating suitable port select signals, an off-board decode circuit for generating a signal used by the wait state generator for inserting a selected number of wait states in the timing thereby adapting off-board memory to on-board operations. The control

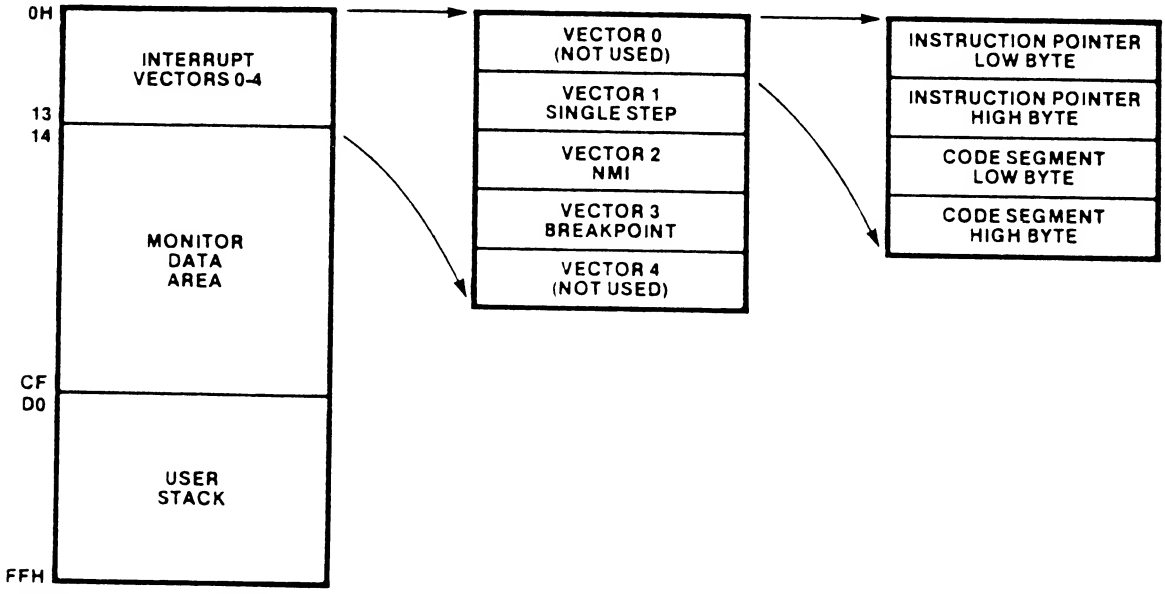


FIGURE 11.3 Intel SDK-86 monitor memory. (Reprinted by permission from SDK-86 User's Guide, *copyright 1978 Intel Corporation.)*

Table 11.2. SDK-86 USART I/O Port Addresses

	Port Function		Port Addresses	$\overline{RD}$	A1
Read	USART	data	FFF0	0	0
Write	USART	data	FFF0	1	0
Read	USART	status	FFF2	0	1
Write	USART	control	FFF2	1	1

transceivers, drivers, and data transceivers are used for such off-board circuit applications.

In the lower left-hand portion of the figure is the serial interface. The serial interface device used is a universal synchronous asynchronous receiver transmitter (USART) that is operated in the asynchronous mode. The baud rate of such a device is selectable by means of jumpers. Interfacing to the USART is achieved by means of two I/O ports in the on-board address space. Such port addresses and their functions are shown in Table 11.2.

In the lower right-hand portion of the figure is the keypad and display section. The basic control device here is the 8279 keyboard display controller. The 8279 functions as a decoder of the keypad matrix as well as a refresher of the display elements.

The keypad itself is shown in Figure 11.4 and a listing of the functions associated with each of the keys is provided in Table 11.3.

The 8-digit display is divided into two groups of four characters each. The group on the left is referred to as the address field, while the group on the right is referred to as the data field. As an example of the operation of the display when the "examine byte" key is pressed, the monitor accepts an entry from the keyboard of a specific address. It should be noted that all memory addresses consist of both a segment value and an offset value. Thus the first address entry is the four-digit segment value, a colon ":" is then entered as a separator, and a second address entry is provided as the offset value. The capacity of an address field is therefore limited to only four characters for either the segment value or the offset value. After such information has been answered the "," key is pressed and the contents of the addressed memory location are displayed on the four-digit display in the data field portion.

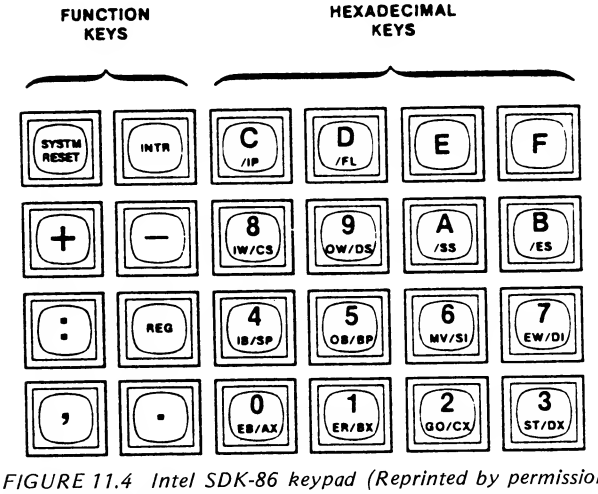


FIGURE 11.4 Intel SDK-86 keypad (Reprinted by permission from SDK-86 User's Guide, *copyright 1978 Intel Corporation.)*

Table 11.3. Example of "Make Versus Buy" Analysis of INTEL SBC 80/10[a]

	Cost ($)				
	10	*50*	*100*	*500*	*1000*
	Number of Boards per Year				
Engineering costs					
Investigation (1.5 E MM)	450	90	45	9	5
Design (1.5 E MM)	450	90	45	9	5
Fabrication (1.0 NE MM)	200	40	20	4	2
Checkout (4.0 E MM)	1200	240	120	24	12
(2.0 NE MM)	400	80	40	8	4
Art work (2.0 NE MM)	400	80	40	8	4
Documentation (1.0 E MM)	300	60	30	6	3
Total	3400	680	340	68	35
Over 3-year life	1133	227	113	23	12
Manufacturing costs					
Tooling (S100)	3	1	–	–	–
PC board	35	25	20	20	20
Board stuffing	10	10	10	10	10
LSI ICs	226	182	142	111	105
Other hardware	55	50	45	40	35
Total	329	268	217	181	170
Test costs	72	72	72	72	72
Total "make" cost	1534	567	402	276	254
Total "buy" cost	395	350	295	265	255

SOURCE: Intel Corporation.

[a]*Engineering, manufacturing and test costs are itemized for a typical single-board computer, the SBC 80/10. To calculate engineering costs, the burdened labor rate was assumed to be $3000 per exempt man month (E MM) and $2000 for non-exempt (NE MM) labor. Total engineering costs are allocated on a per unit basis over a 3-year product life. Manufacturing costs include tooling expense, printed circuit (PC) board cost, hardware cost, and stuffing expense. Test costs are based on testing all boards in a "known good system." No cost is included for automated test equipment. Test time of ¼ hr/board at $30/hr was assumed. For low-volume production, a 50% failure rate was assumed, with 2 hrs required to isolate and correct failure.*

Microprocessor Development Systems

After a system designer has become familiarized with various microprocessors by use of the evaluation systems,

the next step is usually the development of a prototype microcomputer system with a specific microprocessor. Although the user may develop a prototype system from individual components, it is far easier to use the microprocessor development systems commercially available.

A microprocessor development system is a microcomputer specially designed to aid in the development of a prototype hardware system incorporating a specific microprocessor. The microprocessor manufacturers themselves produce such development systems, one of which is the Intel Intellec MDS shown in Figure 11.5. An in-circuit emulator (ice-85) shown in the figure is used to connect the development system to the microprocessor socket on the user's prototype system so that integrated hardware and software may be developed.

The microprocessor development system incorporates various user aids and test programs that enable the prototype system to be fully analyzed.

In addition to the development systems from the microprocessor manufacturers themselves, development systems are also available from a number of systems or instrument manufacturers. The Textronix 8002, shown in Figure 11.6, is one such example that is able to assist in the development of systems incorporating different microprocessors.

Microprocessor development systems have evolved rapidly since their introduction in the mid-1970s. The latest development is a hierarchically distributed processing arrangement of development systems in which one master network control processor controls a plurality of development systems, each of which in turn controls a plurality of slave emulators. Each slave emulator includes a personality module containing the software for the hardware system under test, as well as various test routines that are down-loaded into the emulator from one of the development systems as required. One implementation of such a system, the Futuredata Universal Multiprocessor Development Network, supports the 8080, 8085, Z-80, 6800, and 6802 microprocessors.

Single Board Computers

Although the design of a custom prototype system with the aid of a development system meets the needs of many users with medium- to high-volume requirements, there are many microcomputer applications for which the cost of designing, manufacturing, and testing custom-designed microprocessor systems on printed circuit boards is prohibitive.

Both the semiconductor manufacturers and minicomputer and systems firms saw the need for a general-purpose printed circuit card containing the microprocessor, together with some essential peripheral devices, such as serial and parallel I/O ports, that the user could adapt to his own application and requirements. Although evaluation systems are in fact single board computers, after learning the system the user no longer needs the keypad and display in the final system, and a distinct market for

FIGURE 11.5 Intel Microcomputer Development System (MDS) and in-circuit emulator (ICE). (Courtesy Intel Corporation.)

FIGURE 11.6 Textronix 8002. (Courtesy Textronix Corporation.)

single board computers has developed. In short, the low-volume user is now able to concentrate his efforts on software design and interfacing to his application, while the semiconductor, minicomputer, and system firms are able to concentrate their efforts on providing a wide variety of off-the-shelf computer boards.

One of the first to be introduced was the Digital Equipment Corporation (DEC) LSI-11 in 1975, which was followed the next year by the Intel SBC 80/10 (SBC stands for "single board computer"). As shown in Appendix 2 a wide variety of single board computers and accessory boards now exists.

Even high-volume users may find that the purchase of single board computers from the manufacturer may be cheaper than designing and manufacturing a special-purpose computer board, particularly when one takes into consideration OEM discounts and buying the boards in volume. Such users anticipating demand for more than 100 boards/year may consider a "make versus buy" analysis, such as presented in Table 11.3, a useful exercise.

Logic Analyzers

One of the most important new tools in microcomputer system development is the logic analyzer. A logic analyzer is a test instrument that is designed for analyzing and displaying signal patterns corresponding to logical information (i.e., binary 0 or 1 bits) in either explicit data format (i.e., 1 or 0 sequences of numbers) or in terms of relative voltage levels.

The three basic types of logic analyzers are:

- State analyzers.
- Timing analyzers.
- Trigger generators.

A state analyzer is a device that displays data on signal lines in the form of 1's or 0's on a CRT or similar device. Such devices are particularly useful for analyzing the address, data, and control buses of a microcomputer system. It displays data in the form of a word versus event format.

A timing analyzer is a device that displays information in the form of voltage versus time, much like an oscilloscope.

A trigger generator is a device for use with an oscilloscope to trigger an appropriate signal upon occurrence of a simultaneous signal on two or more input lines.

In addition to logic analyzers, there are a number of less sophisticated products that are used in microcomputer system development, such as a logic probe and an IC test clip.

Logic State Analyzers

The Hewlett Packard 1602A, introduced in 1977 and shown in Figure 11.7, is a typical logic state analyzer. As seen in the figure the unit includes a probe that contacts with the data lines, clock, ground, and other points on the PC card under test. The pushbuttons on the 1602A are then used to define the logic polarity and the edge of the clock pulse being used. Other keys set the display

FIGURE 11.7 Hewlett Packard 1602A logic state analyzer. (Courtesy Hewlett Packard Company.)

of the 1602A to report the information in hexadecimal (as shown in the example), decimal, octal, or binary. Trigger words are selected by the user, and specific message codes appear on the display of the 1602A when a certain event occurs.

The Hewlett Packard 1610A, also introduced in 1977, and shown in Figure 11.8, is an example of a more sophisticated logic state analyzer. The user can trace events by selecting the particular occurrences, coincidences, or logical sequences that are of interest and display the results on the CRT. The operation of the 1610A can be best understood by considering the following description of the function of the control keys, taken from Hewlett Packard literature.

One of the important features of the 1610A is its power to locate trace positions. The analyzer processes the test data at a rate up to 10 MHz in search of the desired trace point and pinpoints that exact location through a series of sequential state conditions set by the operator. From one to seven state conditions, each of which must be encountered by the analyzer in the form and order specified, may be entered into the machine to assure that the analyzer will capture the desired data and not "similar" data.

The desired test parameters are entered in the 1610A through five "menus." The menus offer the user preprogrammed selections and provide variable test data entries for setting up the test and generating a table to present the results. A FORMAT SPECIFICATION key (as shown in Figure 11.8) calls the format menu, which allows the 32 input data channels to be grouped according to labels and allows assignment of logic bases for each label. A TRACE SPECIFICATION key calls the Trace Specification menu with which the operator defines the trace conditions. A Trace may be specified to be a single word or a sequence of from one to seven words that must be found before a trace point is located. The sequence capability allows the 1610A to capture a particular sequence of words located in branches, loops, or nested loops. The 1610A may also be directed to capture ranges of states with an OR specification.

All data are entered in the same label base defined in the Format Specification—octal, hexadecimal, binary, or decimal. The TRACE key causes the 1610A to search for the data; when the analyzer detects that all trace specification conditions are satisfied, the data are captured and displayed on-screen in a 20-word sequence (two control keys allow the data to be "rolled" for viewing all 64 words in memory). Another feature is the TRACE GRAPH key, which directs the 1610A to display a graph of all 64 words in memory. The memory words are presented in a display of data magnitude versus time sequence of occurrence. The graph offers an overview of all 64 words in memory in time-ordered sequence and also shows the level of memory in which a machine is operating. A Trace Compare mode provides an EXCLUSIVE-OR tabular listing between current and stored measurement (entered and interchanged through store keys), displaying identical data as zeroes and different data as nonzeroes.

The 1610A offers three trace modes: Simple, continuous, and compared. Simple tracing conducts one execution of the current measurement. Continuous tracing repeats the execution of a current measurement. Compared tracing, through the TRACE COMPARE key mentioned earlier, conducts a continuous trace until the desired comparison, to determine whether current measurement is equal or not equal with a stored measurement, is achieved.

The 1610A logic state analyzer has nine edit keys. Four control screen cursor movement. DELETE/INSERT keys are used in the Trace Specification menu to enter or remove states within the trace sweep. A default key returns a displayed menu to a known condition. Two graph control keys, INCREMENT and DECREMENT, automatically change upper and lower graph limits.

The correct functioning of the 1610A can be checked by the user through a pushbutton-initiated self-test operation that checks the ROM/RAM, display, keyboard, probes, interrupt, and printer functions—some 99% of the instrument's operations. The pushbutton calls to the CRT screen an instruction program that leads the user through the different test procedures.

The 1610A has a total of 32 high-impedance active probes that operate at rates to 10 MHz. The probes are clustered into four 8-bit pods for system connection and a fifth pod for an external clock source. An important feature of the analyzer is that each pod may be set at different threshold levels, and by format specification, to different word-length formats. Furthermore, the probes are designed with a 0-nsec hold time requirement between clock and data transitions.

Another important logic state analyzer is the Hewlett Packard 1611A introduced in 1976 and shown in Figure 11.9. Unlike the general-purpose 1610A, the 1611A is dedicated to the design and troubleshooting of systems using a particular microprocessor—either the 8080 or 6800. The use of a "personality module," shown in the foreground of Figure 11.9, specializes the 1611A to one type of microprocessor. Many of the features of the 1611A are similar to those of the 1610A, but it would be worthwhile to discuss four of such features in somewhat greater detail:

- Trace specification.
- Display.

FIGURE 11.8 Hewlett Packard 1610A logic state analyzer. (Courtesy Hewlett Packard Company.)

FIGURE 11.9 Hewlett Packard 1611A logic state analyzer. (Courtesy Hewlett Packard Company.)

- Selective trace.
- Counting and timing.

1611A Trace Specification. Trace is specified by entries on the keyboard, determining the conditions that must be met for a trigger to occur. Trace specification may simply be appearance in the data stream of a specified address, or of a specified data word, or the occurrence of a specified combination of 1's, 0's, or "don't cares" on the eight auxiliary probes; or it may be a combination of address, data, and auxiliary-probe conditions. However, instead of such a simple breakpoint trigger, the user may choose ranges of addresses to be used, that is, the trigger specification may not be =, but may be $\geqslant$ or $\leqslant$ limits on the address magnitude. Any or all of the trigger specifications can be further modified by requiring a predetermined "pass count" of up to 256 repetitions before trigger can occur. Finally, trigger ENABLE and DISABLE conditions can be specified, thus establishing the boundary conditions in which the selected window may appear.

The trigger causes a 64-transaction sequence to be stored for display. Trigger can be made to occur at the beginning of the sequence, so the display is of events following the trigger, or it may be at the end, so the display shows events preceding the trigger. Or it may be digitally set anywhere in-between, so the user may study a mixture of before-and-after events in whatever proportion is desired. Digital delay of 1–65,472 memory transactions can be imposed, after other conditions are met, before the instrument begins to trace.

1611A Display. On the 1611A CRT screen appears alphanumeric information about the keyboard settings, as well as the data captured. As directed by the keyboard, the instrument traces and identifies memory transaction after memory transaction in the system's sequence of operations. It stores 64 of these, displaying the top 16 until the scroll keys are used. (The number of instructions identified is usually less than the number of memory transactions, since an instruction often calls for more than one memory fetch.) A switch selects octal or hexadecimal data listing. Op code readout may be numeric, or alphanumeric mnemonics may be spelled

FIGURE 11.10 Hewlett Packard 1615A logic analyzer. (Courtesy Hewlett Packard Company.)

out, as in the microprocessor manufacturer's programming manual.

1611A Selective Trace. Instead of requesting a list of successive transactions, the user may employ the TRACE TRIG key to call for a selective trace of only the occasions when preset trigger conditions are met. For instance, the listing may be of all transactions to a particular address, or all WRITEs to a range of addresses.

1611A Counting and Timing. When counting of events is desired, the user surrounds the trigger with TRIGGER ENABLE and TRIGGER DISABLE conditions and commands a count of the valid triggers occurring between them. The readout is direct, for example, COUNT = 600 EVENTS.

Logic Timing Analyzers

The Hewlett Packard 1615A, introduced in 1977 and shown in Figure 11.10, is an example of a combined logic state and logic timing analyzer for general-purpose use. The 1615A uses a "menu" type of programming as does the 1610A. The "menu" is presented on the keyboard of the 1615A, shown in Figure 11.11, and the user selects one key from each general area of the keyboard, much like a restaurant patron would select one appetizer, one entree, and one dessert from a restaurant menu. The 1615A is capable of tracing events in as many as 24 channels, with events occurring at rates up to 20 MHz.

A memory 24 bits wide and 256 bytes deep in the 1615A can be commanded to capture every event that occurred for 255 clock periods after the trace point of interest, or for 255 periods before, or the trace point information only. The trace point may be positioned anywhere in the 256-word memory. To collect only data of interest, a 6-bit clock qualifier allows selection of only specific data, such as all reads, all writes, and I/O transactions.

The 1615A can be configured three ways with simple

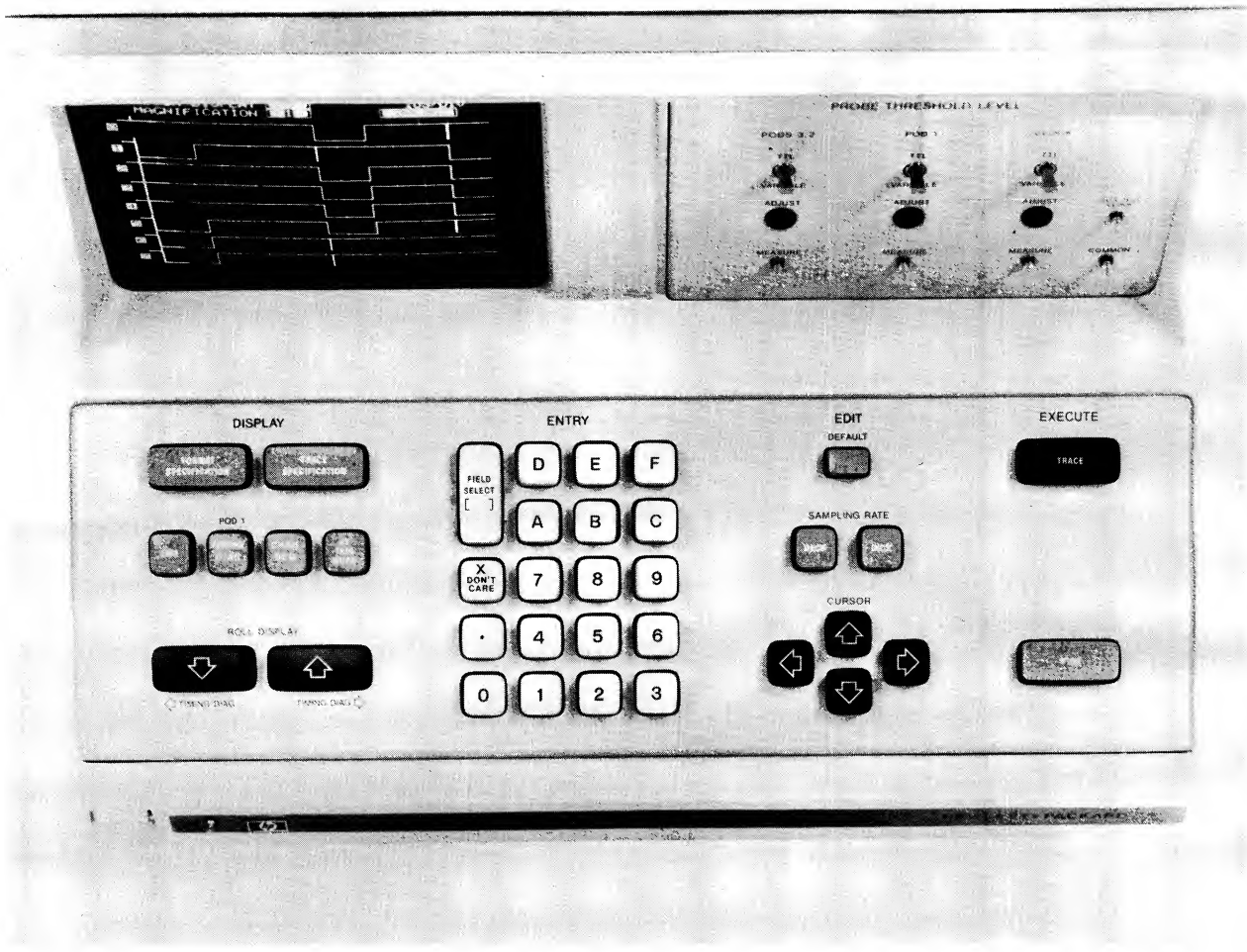


FIGURE 11.11 Hewlett Packard 1615A keyboard. (Courtesy Hewlett Packard Company.)

keyboard entries: as a 24-bit state analyzer 256 words deep, as an 8-bit timing analyzer 256 words deep, or as a combined 16-bit state and 8-bit timing analyzer, each 256 words deep with both operating simultaneously. In the timing mode, glitches are detected and captured, but are only displayed if the user so desires.

The 1615A detects glitches or transient states greater than 5 nsec wide. It not only recognizes glitches, but can separate them from data, and therefore can use these transients as part of a trigger specification. The specification can include both state and/or glitch requirements on any combination of the timing channels. Most important, glitches can be captured during data transitions.

Since both timing and state information are captured simultaneously, the analyzer is capable of capturing time sequences that occur only after a specific state has been accessed, or vice versa. This interactive capability allows the activity on control lines, address decoder outputs, and asynchronous I/O structures to be investigated at specific points in program flow.

In synchronous measurements, the inputs can be configured for hexadecimal, octal, decimal, or binary display formats. They can be identified with three labels for grouping lines as independent variables. In addition, logic polarity can be selected for each label.

For asynchronous measurements, three OR'd trigger fields are available for selecting one or a combination of trigger patterns. In addition, a NOT pattern for triggering can be selected so that status words can be monitored for changes.

Probably one of the most important and difficult measurements to make in any microcomputer system is the capture of a synchronous activity only at a specific point of program execution. For example, it is frequently desirable to look at the conditions of lines on an input port for a few microseconds prior to reading the port to be sure that the data are valid. These interactive measurements can only be accomplished by registering the asynchronous time measurement to a specific state in a synchronous system's program execution, such as

the instruction to "read a port." These measurements require the state and timing analyzers to operate simultaneously and, more importantly, to be interactive in the trigger specifications.

The interaction of both state and time measurements offered by the 1615A opens an entirely new measurement capability, such as identifying what instruction a system was executing at the time of a glitch, verifying I/O line stability just prior to reading a port, and monitoring handshake sequences only at specific points in program execution where problems exist.

Field Test Equipment

The final important type of system development equipment to consider is field test equipment. Field test equipment, such as the Pro-Log M824 System Analyzer shown in Figure 11.12 and the M825 System Analyzer shown in Figure 11.13, are typical examples.

The Pro-Log M824 System Analyzer is a portable instrument that functions as a program monitor for Z80-based microcomputer systems and basically performs the functions of a "front panel" of a microcomputer system. The M825 System Analyzer is similar and operates with 8085-based microcomputer systems. Both units may be packaged in an attache case, as suggested in Figure 11.13, for portability.

The basic features and operations of the units are:

- Display of address, data, machine cycle, and status.
- Static or dynamic display.
- System run or step control.

FIGURE 11.12 Pro-Log M824 System Analyzer. (Courtesy Pro-Log Corporation.)

FIGURE 11.13 Pro-Log M825 System Analyzer. (Courtesy Pro-Log Corporation.)

- System reset.
- Memory I/O address select.
- Address interrupt.
- Data display latch.
- Address stop.
- Interrupt trap and display.
- Dynamic data scan.

The dynamic data scan (DDS) feature permits the user to observe machine operation for a predetermined number of instructions (up to 99, set on the instruction count knobs) past a chosen reference address.

The units interface with the system under test by means of a simple clip-on connector shown in the photographs. A buffer module is also provided to minimize microprocessor system loading to 0.2 mA at 50 pF for address, data, clock, and system control lines.

12

reliability of microcomputer systems

One of the most important concerns of designers of microcomputer systems is reliability. Two aspects of reliability must be considered:

- Device reliability.
- System reliability.

Device reliability is concerned with the reliability of the components used in the system. On this level the user must consider to the following issues:

- Failure mechanisms.
- Reliability testing.
- Quality assurance.
- Operation in severe environments.

System reliability is concerned with the more general issue of system operation and includes such topics as software reliability, a discussion of which is beyond the scope of this book. However, for completeness, we present a brief review of system quality assurance at the end of the chapter.

Device Reliability

Failure Mechanisms

In analyzing device reliability, we begin with a basic review of reliability analysis. The life cycle of a device in

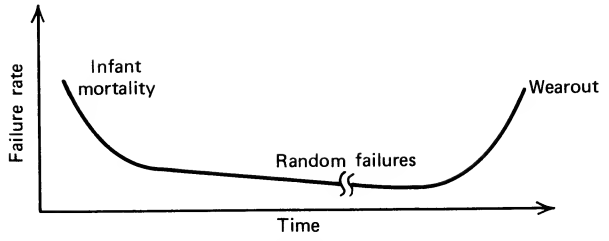


FIGURE 12.1 Reliability life (bathtub) curve.

terms of reliability is often summarized in the curve shown in Figure 12.1, known as the bathtub curve.

The bathtub curve describes the life of a device in terms of its failure rate. The curve can basically be divided into three sequential time periods or phases: infant mortality, random failure, and wearout.

Infant mortality describes the early failures of a device. When a device is produced, there are a number of possible failure mechanisms that are not immediately evident but that exhibit themselves under moderate stress or operational levels. Such failures therefore occur early in the life cycle of the device, and hence the term "infant mortality."

The next period is the "random failure" portion of the curve. After a moderate period of time (typically in the low hundreds of hours), the failure rate reaches a low value, and only occasional random failure mechanisms occur. During the random failure portion of the life cycle, there is a relatively small decline in the failure rate as a result of the gradual depletion of potential random failures from the general population.

Finally, the wearout phenomena occurs at the end of the device's lifetime. As the curve indicates, there is a rapidly rising failure rate.

There are specific failure mechanisms for each of these periods in the device's life cycle. Table 12.1 gives some of the common failure mechanisms associated with an MOS device, along with the portion of the life cycle curve it affects, the associated thermal activation energy, and the test method used for detection.

It is also worthwhile to consider the relative importance of these failure mechanisms. Table 12.2 illustrates a typical failure mode distribution for integrated circuits, neglecting various anomalous defects detected by a standard screening procedure. More details on the various types of failures are presented later in this chapter.

In considering failure mechanisms in more detail, we turn our attention to the following issues:

- Random failure.
- Distribution of sample failure rates.
- The Arrhenius equation.
- Wearout phenomena.

Table 12.1. Failure Mechanisms In MOS Devices

Failure Mode	*Type*	*Activation Energy* (E_a)(eV)	*Detection*
Slow trapping	Wearout	1.0	High temp. bias
Contamination	Wearout/ infant	1.4	High temp. bias
Surface charge	wearout	0.5-1.0	High temp. bias
Polarization	Wearout	1.0	High temp. bias
Electro-migration	Wearout	1.0	High temp. operating life
Microcracks	Random	—	Temp cycling
Contacts	Wearout/ infant	—	High temp. operating life
Oxide defects	Infant/ random	0.3	High Voltage Operating life and cell stress
Silicon defects	Infant/ random	0.3	High voltage cell stress
Electron Injection	Wearout	—	Low temp. high Voltage operating Life

SOURCE: Intel Corporation. Reprinted with permission.

Table 12.2. Failure Mode Distribution (%)

Surface related	39
Package	26
Wire defects	10
Bulk silicon	7
Metalization	7
Oxide	6
Bond	5

Both the infant mortality and random failure portions of the bathtub curve can be described by the probability

$$P_o = e^{-\lambda t}$$

which expresses the probability of having no failures at a specific time. Here λ is the failure rate, and t is time. The frequently used value "mean time between failures" is equal to $1/\lambda$.

Since we are concerned with samples of the entire population of devices, a curve such as that shown in Figure 12.2 is used to express the distribution of failure rates, which is known as a normal distribution. The mean of the distribution expresses the best estimate for the failure rate λ.

It should further be noted that temperature accelerates failures, with the behavior fitting the Arrhenius equation:

$$R = R_0 e^{-\theta/kT}$$

where θ is the activation energy in electron volts, k is

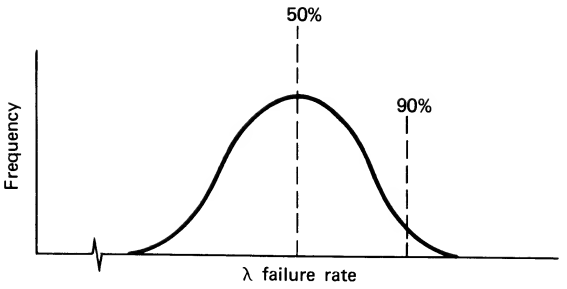


FIGURE 12.2 Distribution of sample failure rates.

Boltzman's constant, and T is the temperature in degrees Kelvin. Use of the Arrhenius equation is made in reliability testing, as is explained in the next section.

Finally, the wearout phenomenon differs from the failure modes described above. Wearout may occur through electromigration of circuit metallization or through electrolytic corrosion in nonhermetic devices (e.g., devices in plastic packages).

Reliability Testing

The following test methods are used to assure electrical reliability of a device:

- High temperature dynamic life test.
- Dynamic life test at 70°C.
- High-temperature reverse-bias.
- Device stability.
- Low-temperature life test.
- High-temperature storage.
- Temperature cycling.

A description of these methods, as used by Intel Corporation in testing the reliability of their HMOS static RAM family, follows.

"*High-Temperature Dynamic Life Test.* This test is used to accelerate failure mechanisms by operating the devices at an elevated temperature. For static RAMs the operating temperature is 125°C. The data obtained are translated to a lower temperature using the Arrhenius plot in Figure 12.3, giving a larger number of equivalent hours of test.

"During the test the memory is sequentially addressed and filled with alternating patterns of 1's and 0's. The outputs are exercised, but not monitored or loaded. Figure 12.4 shows timing and connection diagrams for this test on the HMOS static RAMs. To best determine long-term failure rates all devices used for all life testing (high-temperature, HTRB, and system) are subjected to standard production screens plus a 48-hr burn-in to weed out infant mortality.

"*70°C Dynamic Life Test.* This test is used to duplicate operating conditions of units at maximum data sheet conditions. The data obtained are used to determine validity of high temperature results. Unit-hours accumulated here are combined with other tests to determine the failure rate. Bias diagram and system operation are the same as high temperature dynamic life test. Results are summarized in Table 12.3 with other life test data.

"*High-Temperature Reverse Bias (HTRB).* This test is performed to detect failure mechanisms (see Table 12.1), which are accelerated by high temperature (150°C). This test is effective in accelerating leakage-related failures and drifts in device parameters due to process instability. The HTRB bias diagrams for HMOS static RAMs are shown in Figure 12.5. High-temperature reverse-bias results are included in the life test summary.

"*Device Stability.* Since the HMOS static RAM family operates in high-speed memory systems with critical timing, the stability of device parameters is essential. To ensure there is no drift in critical parameters, life tests are conducted on single transistors as well as on actual RAMs.

"A good measure of wafer technology stability is the threshold voltage of a single MOS transistor. Transistors are available as test structures on every wafer to monitor process parameters. The transistors were assembled and life tested under static bias shown in Figure 12.6. These life tests were conducted at 160°C to insure there is no shifting of threshold voltage due to slow trapping or contamination.

"The access time of a memory is also a good measure of its performance. The stability of the HMOS static RAM family was measured using access time versus V_{CC} as a measure of device performance before and after standard life testing.

"*Low-Temperature Life Test.* This test is performed at maximum operating frequency to detect the effects of electron injection into the gate oxide. The conditions for electron injection occur during transitions when the transistors are in saturation. This test is performed at -20°C in a bath of Flourinert FC-78 to obtain maximum cooling. Higher than normal power supply voltage can be used to accelerate this effect. A life test at V_{CC} = 7 V increases hot electron injection currents by a factor of 500 over normal 5 V operation. Access time measurements are used to insure that there is no degradation.

"*High-Temperature Storage.* Another common test is high-temperature storage, in which devices are subjected to elevated temperatures (160°C for plastic packages and 250°C for hermetic packages) with no applied bias. This test is used to detect mechanical reliability problems (e.g., bond integrity) and process stability.

"*Temperature Cycling.* This test consists of cycling the

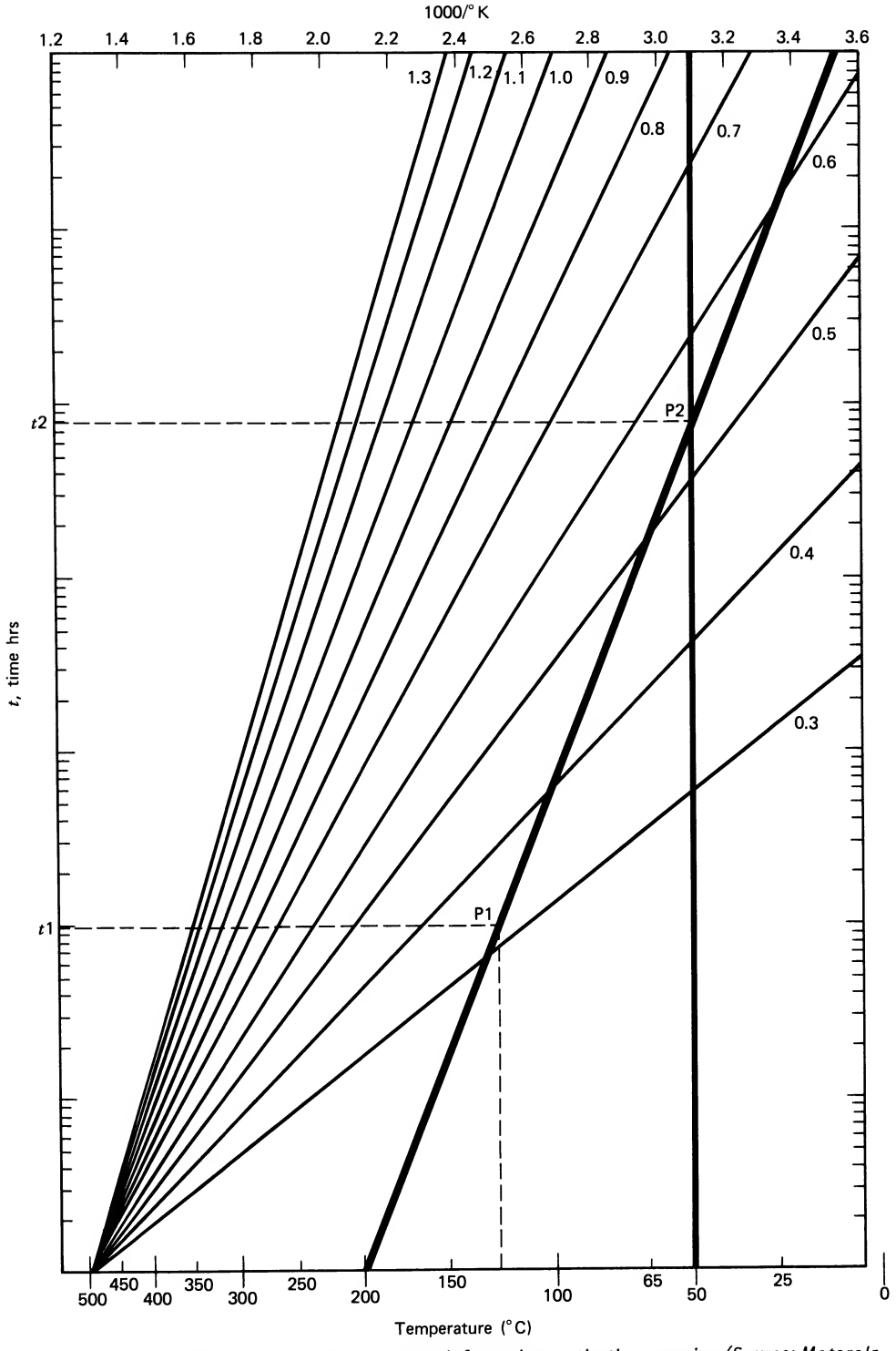


FIGURE 12.3 Arrhenius time-temperature graph for various activation energies. (Source: Motorola. Courtesy Motorola Semiconductor Products, Incorporated.) For increased flexibility in working with a broad range of device hours, the time-temperature regression lines have been normalized to 500°C and the time scale omitted, permitting the user to define the scale based on his own requirements.

TABLE 12.3. High-Temperature Life Test Results[a]

Month	Device	T_A (°C)	168 Hrs	500 Hrs	1K Hrs	3K Hrs	5K Hrs
Pre-1977	PE2125A	125	0/349	0/349	0/349	0/99	0/99
	DD2125A	125	1/788 A	1/619 C	1/618 B	0/62	
	DD2115A	125	1/139 D	0/138	0/138	0/99	0/99
Jan '77	PE2125A	125	1/210 E	1/100 F	0/99		
Feb '77	PE2125A	125	1/210 G	0/199	1/199 H		
	DD2115A	125	0/84	0/84	0/84		
Mar '77	DD2115A	125	0/84	0/84	0/84		
Apr '77	DD2125A	125	0/84	0/84	0/84		
May '77	PE2125A	125	0/48	0/48	0/48		
	DD2125A	125	0/84	0/84	0/84		
Jul '77	DD2125A	125	0/84	0/84	0/84		
Aug '77	PE2125A	125	0/84	0/84	0/84		
	DD2125A	125	0/200	0/200	0/200		
TOTAL	2115A/25A	125	4/2448	2/2157	2/2155	0/260	0/198
May '77	CL2147	125	0/100	0/100	0/100	0/60	0/60
Jun '77	CL2147	125	0/100	0/100	0/100	0/60	1/60 I
Aug '77	CL2147	125	0/78	0/78	0/78		
Sep '77	CL2147	125	0/1962				
Oct '77	CL2147	70	0/300	0/300	0/300		
Oct '77	CL2147	125	0/657	0/657	0/657		
Oct '77	CL2147	150*	0/78	0/78	0/78		
Dec ' 77	CL2147	125	0/277	0/277	0/277		
TOTAL	CL2147		0/3552	0/1590	0/1590	0/120	1/120
Nov '77	PE2141	125		0/105	0/105		
Nov '77	PE2141	150*		0/25	0/25		
Dec '77	PE2141	125		0/306	0/306		
Dec '77	PE2141	150*		0/50	0/50		
Jan '77	DD2141	125		0/87	0/87		
TOTAL	2141			0/573	0/573		

*HTRB

A Single bit fail at low V_{CC}, contamination, 1.0 eV
B Single bit fail CKBD pattern, silicon defect in isolation region, 0.3 eV
C Single bit, oxide breakdown, 0.3 eV
D Single bit, 1.0 eV
E Multiple edge bits fail, contamination, 1.0 eV
F Column failure, contamination, 1.0 eV
G Single bit, fails high V_{BB} (more negative), threshold drift, 1.0 eV
H Pattern sensitive column failure, contamination, 1.0 eV
I Single bit fails high V_{CC}, silicon defect in pulldown transistor, 0.3 eV

[a] *Source: Intel Corporation. Reprinted by permission of Intel Corporation, copyright 1978. All mnemonics copyright Intel Corporation 1978.*

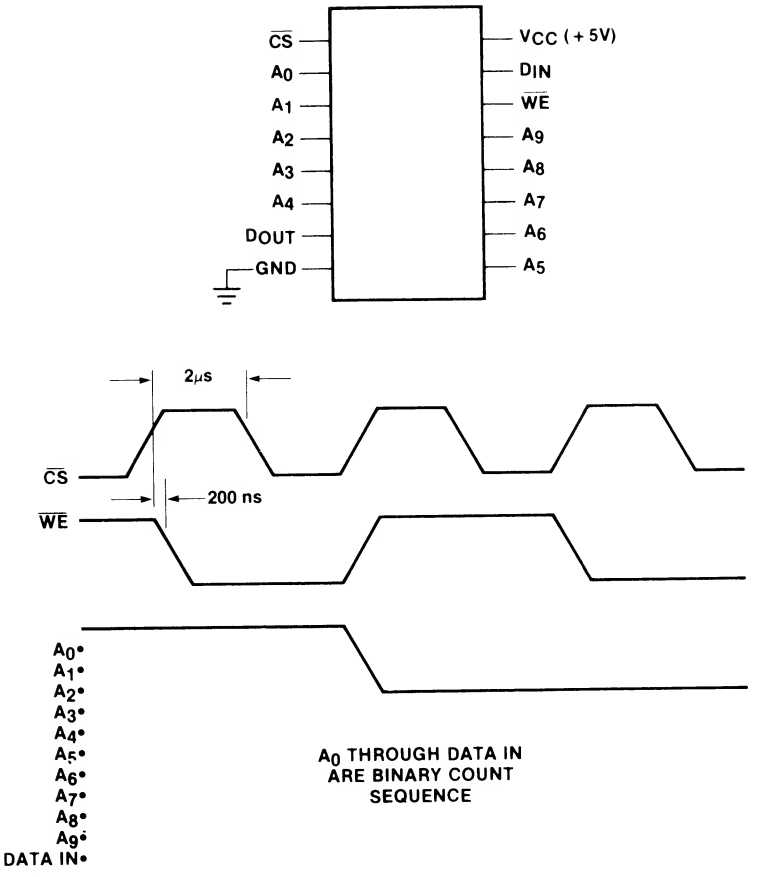
temperature of the chamber housing the devices from -65 to 150°C. This test is used to detect mechanical reliability problems and microcracks."

Quality Assurance

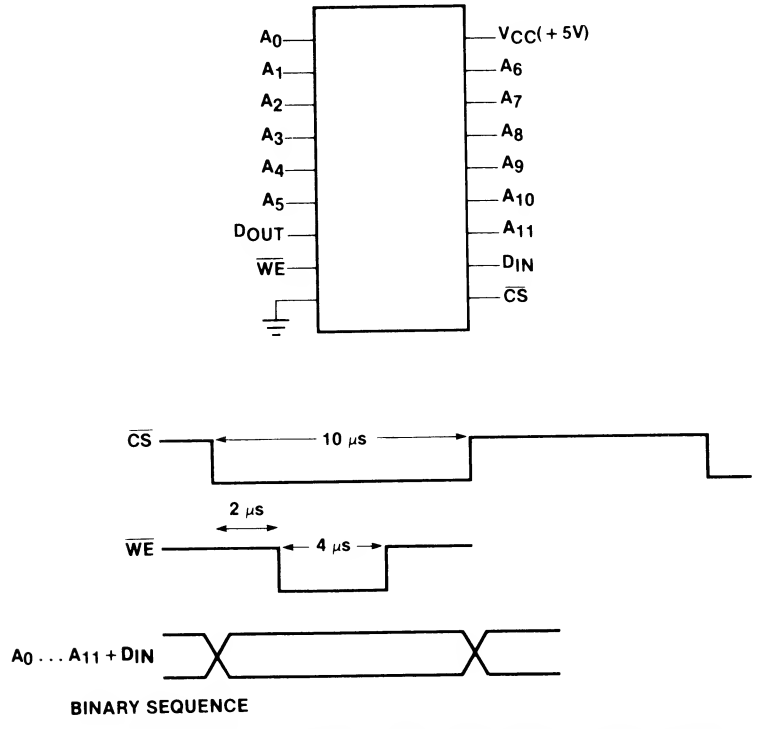
Quality assurance is concerned with setting up criteria to determine how well a lot of devices meets the quality and reliability requirements of a customer or application. There are basically three methods:

- One hundred percent testing using a percent defective allowable (PDA).
- Sampling based on an acceptance quality limit (AQL).
- Sampling based on a lot tolerance percent defective (LTPD).

The PDA method establishes a predetermined percentage level of rejections above which the entire lot will be rejected. Since every device is tested, this is a 100%

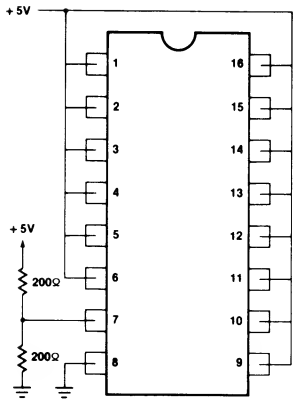


(a) 2115A/2125A Dynamic Life Test Bias and Timing



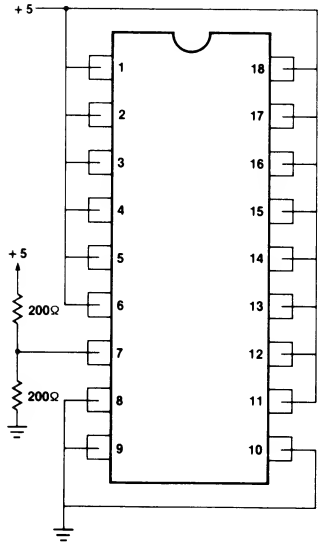
(b) 2141/2147 Dynamic Life Test Bias and Timing

FIGURE 12.4 Dynamic life test bias and timing (Source: Intel Corporation.) Reprinted by permission on Intel Corporation, copyright 1978. All mnemonics copyright Intel Corporation 1978.



PIN	PIN NAME	STRESS
1	$\overline{CS}$	+5V
2	A_0	+5V
3	A_1	+5V
4	A_2	+5V
5	A_3	+5V
6	A_4	+5V
7	D_{OUT}	Res. LD
8	GND	GND
9	A_5	+5V
10	A_6	+5V
11	A_7	+5V
12	A_8	+5V
13	A_9	+5V
14	$\overline{WE}$	+5V
15	D_{IN}	+5V
16	V_{CC}	+5V

(a) 2115A and 2125A



PIN	PIN NAME	STRESS
1	A_0	+5V
2	A_1	+5V
3	A_2	+5V
4	A_3	+5V
5	A_4	+5V
6	A_5	+5V
7	D_{OUT}	Res. LD
8	$\overline{WE}$	GND
9	GND	GND
10	$\overline{CS}$	GND
11	D_{IN}	+5V
12	A_{11}	+5V
13	A_{10}	+5V
14	A_9	+5V
15	A_8	+5V
16	A_7	+5V
17	A_6	+5V
18	V_{CC}	+5V

(b) 2141 and 2147

FIGURE 12.5 High-temperature reverse-bias diagrams. (Source: Intel Corporation.) Reprinted by permission of Intel Corporation, copyright 1978. All mnemonics copyright Intel Corporation 1978.

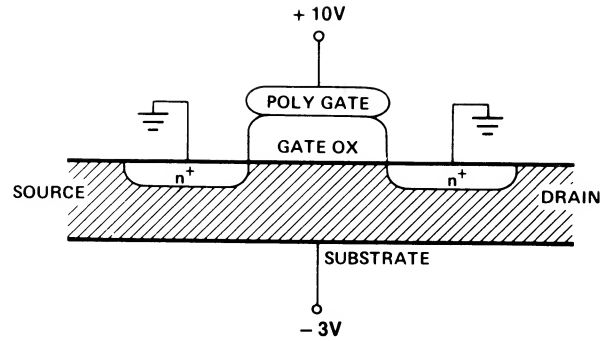


FIGURE 12.6 High-temperature bias of single transistor. (Source: Intel Corporation.) Reprinted by permission of Intel Corporation, copyright 1978. All mnemonics copyright Intel Corporation 1978.

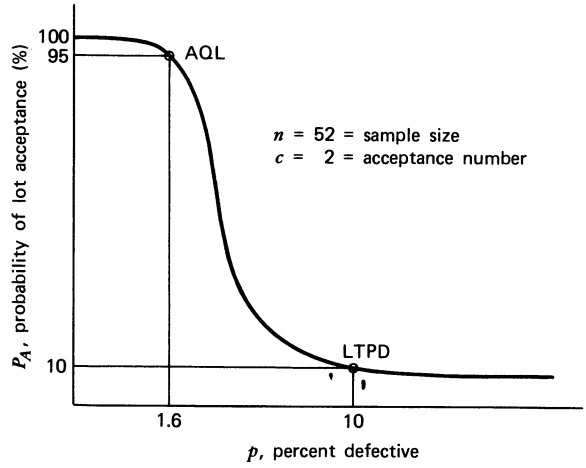


FIGURE 12.7 Operating characteristic curve. (Source: Motorola. Courtesy Motorola Semiconductor Products, Incorporated.)

testing method and is generally time-consuming and expensive.

The AQL method establishes 95% as the probability of lot acceptance based on the sampling procedure. The sample size is variable depending on the lot size so that the percentage probability objective is maintained. Another way of looking at the percentage probability is that the risk of rejecting a good lot is 5%, or 1 chance in 20.

The LTPD method establishes a probability of 10% of the consumer accepting a bad lot, assuming that the devices contains percent defective p.

A good way to illustrate the difference between the AQL and the LTPD method is to plot the probability of lot acceptance (in percent) versus the percent defective. Such a curve is called the operating characteristic curve and is shown in Figure 12.7. At a very small percent defective, the probability of lot acceptance is at or around 100%. At a very high percent defective (e.g., 10%), the probability of lot acceptance is very low (e.g., 10%). In the present example, the AQL point is marked on the graph and a percent defective of 1.6% is seen to correspond with such criteria. Similarly, the LTPD point is marked, and a percent defective of 10% corresponds to such criteria.

Operation in Severe Environments

Another aspect of the reliability of devices is the issue of operation in severe environments beyond merely accelerated pressure, temperature, humidity, and bias conditions, for example, radiation environments. In ad-

dition to the obvious military applications, such semiconductor devices capable of withstanding high levels of radiation are important in applications such as nuclear power generating stations, radiation monitoring and detection equipment, and space applications.

Exposure to radiation introduces various structural defects in the semiconductor material, thereby changing their electrical characteristics and thus also the operation of the circuit. Such changes are dependent both on the duration and type of radiation and on the specific resistivity, impurity type and concentration, temperature, and bias or carrier injection level of the semiconductor material. There are three basic types of radiation effects to consider:

- Steady state ionizing radiation.
- Transient ionizing radiation.
- Neutron radiation.

Steady state ionizing radiation produces an accumulation of charges that is a function of the total absorbed dose. Total absorbed dose is usually measured in rads (Si), which are units of roentgens absorbed dose in silicon. Such radiation causes significantly increased leakage current in bipolar devices and significant changes in threshold voltages in MOS devices, as well as other changes in various device parameters. The magnitude of such changes of device parameters varies with the specific type of MOS device (*p*-channel, *n*-channel, or complementary MOS), as well as the voltage bias present in the circuit at the time of radiation exposure. Figure 12.8 shows the effect of radiation on the threshold voltage for both *n*-channel and *p*-channel MOS devices as a function of the total ionizing radiation dose. The figure also shows such changes under different bias conditions of the device.

Figure 12.9 shows the drain leakage current that occurs in an *n*-channel device as a function of the total ionizing radiation dose, as well as that for different voltage bias conditions.

Transient ionizing radiation refers to the transient upset of circuit operation due to a short pulse of radiation. In addition to temporary shifts of MOS device parameters similar to those associated with the permanent degradation as a result of steady state ionizing radiation, there are radiation generated photocurrents in reversed-bias *pn* junctions, causing changes of the logic state in bipolar digital circuits as well as the phenomena of "latch-up" in MOS circuits. In a latch-up, a device latches in one state as a result of silicon controlled rectifier action or second breakdown and remains there indefinitely. The latchup phenomenon generally varies depending on the particular fabrication process used.

There are various methods for improving the tolerance of devices to the effects of ionizing radiation. Such techniques are known as "radiation hardening" and in-

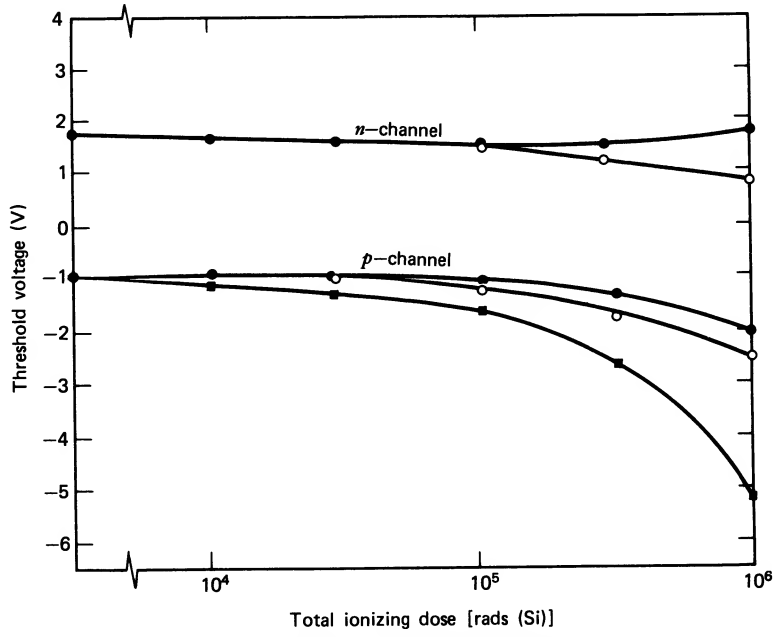


FIGURE 12.8 Effect of Radiation on threshold voltages for different bias conditions. (Source: T. P. Haraszti, IEEE JSSC Vol. SC-13, No. 5, 1978.) Voltage bias for n*-channel: (●)* V_{DS} = +10 V, (○) V_{GS} = *+10 V; for* p*-channel: (●)* V_{GS} = *-10 V, (○)* V_{DS} = *-10 V, and (■)* V_{GS} = *+10 V.*

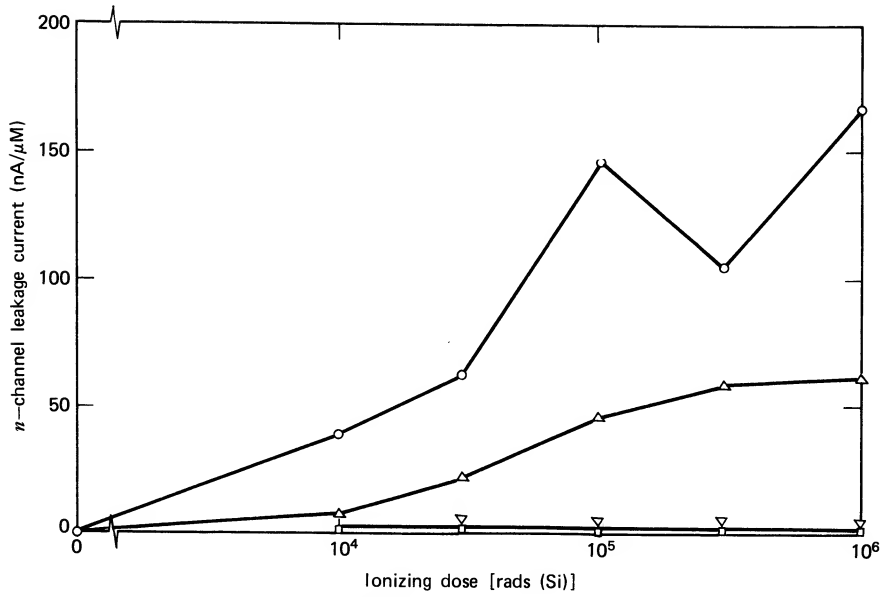


FIGURE 12.9 N-channel drain leakage as a function of radiation dose and voltage bias. (Source: T. P. Haraszti, IEEE JSSC Vol. SC-13, No. 5, 1978.) Voltage bias: (○) V_{DS} = +10 V, (△) V_{GS} = +10 V, (△) V_{DS} = V_{GS} = 0 V, and (□) V_{GS} = -10 V.

clude both special manufacturing techniques as well as circuit design modifications. It must be realized however, that such changes also impact the component density, speed, power, chip size, and yield of such devices.

Neutron radiation permanently degrades the gain in both bipolar and MOS devices and increases the saturation voltage of bipolar transistors.

In general, neutron radiation does not have that great an effect on semiconductor circuits. Only at levels as high as 10^{10}-10^{12} neutrons/cm^2 do devices begin experiencing change in electrical characteristics. MOS circuits in particular are not susceptible to neutron radiation below levels of about 10^{10} neutrons/cm^2.

System Reliability

System reliability is concerned with the more general problem of the entire system operation under both normal and adverse conditions. Although device reliability is an important aspect of system reliability, there are some additional factors that contribute to system reliability that are not present at the device level. These factors include:

- Cross talk between adjacent conductors.
- Noise.

Cross talk between adjacent conductors is due to capacitive coupling between the conductors, which results in unwanted signals being generated in the adjacent conductor. Although cross talk can occur at the device level, it is generally considered by the device designers and minimized by circuit or layout techniques. Cross talk between devices is more of a problem for the system designer who is laying out a printed circuit card in which various signal lines lie close to one another, and especially when such lines extend parallel to one another.

Noise is a more general term for an undesired signal. Although cross talk is actually a form of noise, for the purpose of our analysis noise is generally considered to be white or spectrumless.

There are a number of different noise sources in any one system, depending on the particular configuration. Examples of noise sources include:

- Input noise.
- Thermal noise.
- Amplifier noise.
- Transfer inefficiency noise.

Input noise is the noise associated with the input signal to the system. It may be due to thermal fluctuations in the signal itself, or it may be noise produced in a process of acquiring the input signal, such as quantitizing or sampling errors. The spectrum of input noise is generally uniform.

Thermal noise is the noise associated with thermal fluctuations of signals internal to the system. Such noise

is sometimes referred to as Johnson noise. The spectrum of thermal noise is also uniform.

Amplifier noise is associated with the amplifier stage of the system. In addition to the noise associated with the voltage or current gain produced by the amplifier stage, noise also can be introduced from leakage currents or reset operations. While amplifier noise is generally uniform, reset noise can be partially eliminated by suitable sampling procedures.

Finally, transfer inefficiency noise is a correlated noise with a maximum at the Nyquist frequency and has a predetermined spectrum.

In many devices only certain of these noise sources are important and others are relatively small. Information on noise can be obtained by analyzing the spectrum signatures of noise and determining whether the noise is correlated, white, $1/f$, or some other function. Such noise spectral information is of great importance for optimizing a performance of a system or group of devices.

To determine the amount and nature of the noise, two approaches are necessary. First, consideration must be given to the amplitude variation in the output signal, such as might be seen on an oscilloscope. Second, consideration must be given to various types or sources of noise as might be analyzed with a spectrum analyzer. In the first case, one might collect a large sample of output values and statistically obtain their mean and deviation. The second approach would be to use a spectrum analyzer for measuring the noise across a predetermined band such as from 0 to 50 MHz. In some cases the result of the spectrum analyzer may indicate that the noise has a certain value across the entire spectrum, while at frequencies below 100 KHz, for example, a slight rise in the noise level might be present. Such a change in noise level might indicate a $1/f$ effect. One can then integrate the noise contributions over the entire spectrum and obtain a value in millivolts for the total noise.

System Quality Assurance

Now that system reliability is understood, we can turn our attention to system quality assurance. There are a number of different approaches:

- Self-diagnosis and correction.
- External error detection/correction.
- Redundant systems.

Self-Diagnosis and Correction

Self-diagnosis is the simplest and least expensive approach to system quality assurance. Self-diagnosis is the technique of providing some means within the system itself for testing devices or functions. Such means may include hardware, software, or both.

An example of self-diagnosis is to provide internal parity checking of memory words. The detection of a parity error would indicate that a particular memory word has been added, dropped, or modified in error. Parity checking is an operation that may be performed either by hardware or software.

Another technique of self-diagnosis is through software. Diagnostic programs can be incorporated in the system and run routinely by the operating system at periodic maintenance intervals for providing automatic self-testing of the system, or they can be run at specific times such as power-on or reset. Alternatively, diagnostic programs can be stored externally and run only when necessary, such as when other indicators suggest an error or system fault.

Diagnostic programs may be used to test many different parts of the microcomputer system, including:

- Microprocessor.
- Memory.
- Programmable I/O ports.
- Peripheral devices.

The user must determine which elements of the system are the most failure prone and direct the testing efforts to those units. It would be worthwhile to review some of the important testing issues for each of these elements.

Testing of the microprocessor is concerned with testing the operation of each individual instruction, as well as the operation of each functional unit of the microprocessor. It is generally not necessary to test every instruction of the microprocessor's instruction set, but only those that are used by a program, the most frequently used instructions, or the instructions that are most likely to fail. (An instruction that is dependent on many separate factors would be more likely to fail than one that depends on the correctness of fewer variables.)

It should be noted that this discussion of testing of the microprocessor is concerned with hardware faults. The subject of software faults and software reliability is not covered here.

Testing of the memory is concerned with testing each individual memory cell for correct read and write operations, as well as testing groups of cells.

Testing of the programmable I/O ports and peripheral devices is similar to testing of the microprocessor and memory.

External Error Detection/Correction

External error detection/correction is the use of external test equipment for monitoring system operations, de-

termining if an error has occurred, and correcting such error.

Redundant Systems

Another method of system quality assurance is through the use of more than one system performing the same operation simultaneously, or providing a standby "back-up" system ready to assume processing responsibility if a failure should occur in the main system.

References

HMOS Reliability, Intel Corporation, Reliability Report RR-18, Santa Clara, Calif., 1978.

Memory/LSI Data, Reliability Analysis Center, Publication MDR-13, Griffiss AFB, New York, 1979.

MIL-M-38510A, General Specifications for Microcircuits, 1972.

MIL-STD-883A, Test Methods and Procedures for Microelectrics, 1974.

APPENDIX

1

microprocessors and microprocessor manufacturers

Manufacturer	*Designation*	*Technology*
	1-Bit Microprocessor	
Motorola	MC14500B	CMOS
	4-Bit Microprocessors	
AMI	S2000	NMOS
Intel	4004	PMOS
	4040	PMOS
National	COPS: 402	
	404	
	5782	PMOS
	5799	PMOS
	57140	PMOS
NEC	555	
	556	

Manufacturer	*Designation*	*Technology*
	4-Bit Microprocessors (*Continued*)	
Panasonic	MN 1400	NMOS
	MN 1430	PMOS
	MN 1450	CMOS
	MN 1498	
	MN 1499	
	MN 1599	
Rockwell	PPS 4/1: MM75,76,77 78	PMOS
	PPS 4/2	PMOS
Sharp	SM-4	CMOS
Texas Instruments	TMS-1000	PMOS
	1100	PMOS

Manufacturer	*Designation*	*Technology*
4-Bit Microprocessors (*Continued*)		
Texas Instruments (*Cont.*)		
	1200	PMOS
	1300	PMOS
Western Digital	1872	PMOS
8-Bit Microprocessors		
Electronic Arrays	9002	NMOS
Fairchild	3850 (F-8)	NMOS
	3859	NMOS
	3860	NMOS
General Instruments	LP-8000	PMOS
	PIC-1650	NMOS
	PIC-1655	NMOS
Intel	8008	PMOS
	8080	NMOS
	8035/8039	NMOS
	8041/8741	NMOS
	8048/8748	NMOS
	8085	NMOS
	8088	NMOS
MOS Technology	650X	NMOS
	651X	NMOS
	660X	NMOS
Mostek	3870	NMOS
	3871	NMOS
	3874	NMOS
Motorola	6800	NMOS
	6801	NMOS
	6802/6808/6846	NMOS
	6803	NMOS
	6805	NMOS
	6809	NMOS
National	ISP8A600 (SC/MP-II)	NMOS
	INS 8040	NMOS
	INS 8060	NMOS
	INS 8070	NMOS
	16008	NMOS
RCA	1802	CMOS
	1803	CMOS
Rockwell	PPS-8	PMOS
	PPS-8/2	PMOS
Telefunken	CP3-F	PMOS
Toshiba	T3444	NMOS
Western Digital	1621	NMOS
Zilog	Z80	NMOS
	Z8	NMOS
12-Bit Microprocessors		
Intersil	6100	CMOS
Toshiba	3190	

Manufacturer	*Designation*	*Technology*
16-Bit Microprocessors		
Advanced Micro Devices	29116	Bipolar
Data General	mN 601	NMOS
	mN 602	NMOS
Fairchild	9440	Bipolar
	9445	Bipolar
Ferranti	F100L	CDI Bipolar
General Instruments	CP-1600/1610	NMOS
Hewlett Packard	MCC	SOS
Intel	8086	NMOS
Motorola	68000	NMOS
National	PACE	PMOS
	INS 8900	NMOS
	NS 160016	NMOS
	NS 160032	NMOS
Panafacom	PFL-16A	NMOS
Texas Instruments	TMS9980/9981	NMOS
	TMS9900	NMOS
	SBP9900	I^2L
	TMS9940	NMOS
Western Digital	WD-16	NMOS
	PASCAL Microengine	NMOS
Zilog	Z8000	NMOS
Bit-Slice Microprocessors		
Advanced Micro Devices	2901	Schottky bipolar
	2903	Schottky bipolar
Fairchild	9405	Schottky bipolar
	34705	CMOS
	100220	ECL bipolar
Intel	3002	Schottky bipolar
Motorola	10800	ECL bipolar
Texas Instruments	SBP0400	I^2L
	SN54S/74S-481	Schottky bipolar

Microprocessor Manufacturers

Advanced Micro Devices
901 Thompson Place
Sunnyvale, CA 94086
(408) 732-2400

American Microsystems, Inc.
3800 Homestead Road
Santa Clara, CA 95051
(408) 246-0300

Electronic Arrays, Inc.
550 Middlefield Road
Mt. View, CA 94043
(415) 964-4321

EM&M SEMI
3883 N. 28th Avenue
Phoenix, AZ 85017
(602) 263-0202

Microprocessor Manufacturers (*Continued*)

Fairchild Semiconductor
464 Ellis Street
Mt. View, CA 94040
(415) 962-5011

Fujitsu America Inc.
2945 Oakmead Village Court
Santa Clara, CA 95051
(408) 985-2300

General Instrument Corp.
600 W. John Street
Hicksville, NY 11802
(516) 733-3000

Harris Semiconductor
Box 883
Melborne, FL 32901
(305) 727-5407

Intel Corp.
3065 Bowers Avenue
Santa Clara, CA 95051
(408) 987-8080

Intersil, Inc.
10900 N. Tantau Avenue
Cupertino, CA 95014
(408) 257-5450

Monolithic Memories, Inc.
1165 E. Arques Avenue
Sunnyvale, CA 94086
(408) 739-3535

MOS Technology, Inc.
950 Rittenhouse Road
Norristown, PA 19401
(215) 666-7950

Mostek
1215 W. Crosby Road
Carrollton, TX 75006
(214) 242-0444

Motorola Semiconductor Pro. Div.
5005 E. McDowell Road
Phoenix, AZ 85008
(602) 244-6900

National Semiconductor Corp.
2900 Semiconductor Drive
Santa Clara, CA 95051
(408) 737-5000

NEC Microcomputers, Inc.
173 Worcester Street
Wellesley, MA 02181
(617) 237-1910

Microprocessor Manufacturers (*Continued*)

Panasonic Electronic Components
One Panasonic Way
Secaucus, NJ 07094
(201) 348-7269

Raytheon Semiconductor Div.
350 Ellis Street
Mt. View, CA 94042
(415) 968-9211

RCA Solid State Div.
Route 202
Somerville, NJ 08876
(201) 722-3200

Rockwell International
3310 Miraloma Avenue
Anaheim, CA 92803
(714) 632-6650

Sharp Electronics
10 Keystone Place
Paramus, NJ 07652
(201) 265-5600

Solid State Scientific, Inc.
Montgomeryville, PA 18936
(215) 855-8400

Texas Instruments, Inc.
P.O. Box 5012
Dallas, RX 75222
(214) 238-4881

Western Digital Corp.
19242 Red Hill Avenue
Newport Beach, CA 92663
(714) 557-3550

Zilog Inc.
10460 Bubb Road
Cupertino, CA 95014
(408) 446-4666

APPENDIX

2

single board microcomputer systems*

Reprinted with permission from Electronic Design, *Vol 27, No. 5, March 1, 1979, copyright Hayden Publishing Co., Inc., 1979).*

					Memory Kbytes (K-1024)						Serial I/O	
Manufacturer	*Model*	*Word Size in Bits (data/address)*	*CPU Type*	*Clock Freq. (MHz) Min./Max.*	*Total Addressable*	*Amount of RAM on Card (b=bytes)*	*Amount of ROM on Card*[a]	*DMA Capability*	*Bus Type (P-proprietary, blank-no bus)*	*Parallel I/O Lines*	*Number of Ports*	*Baud Rate (max.) (k Baud)*
Advanced Micro Computer	SKC 85	8/16	8085A	3	64	1.28	0/4		P	44	1	
American Microsystems	EVK 300	8/16	6800	1	16	1	6	•		56	1	9.6
Analog Precision	TPM2	8/16	8080A	1.1/2	32	0.25	1.2		P			
Apple Computer	A2B0004X	8/16	6502	1.023	64	4/48	0/12		P		1	9.6
Applied Systems	ASC/80	8/16	8085	0.5/5	64	0.25/1	4		P		1	
Compas Microsystems	CSB-1.2	8/16	6502	1/2	64	2	0/8		System 65	50/40	0/1	9.6
Computer Automation	LS14/10	16/16	Custom	16	128	1/4 kwds		•	P	32		19.2
Control Logic	CSS-1143	8/16	Z-80	2	64	1	0/16		P		1	110b
	MM1-MSC	8/16	Z-80	2/4	64	1024b	4				4	50
Creative Microsystems	CMS 9600	8/16	6802	0.5/1	56	1.2	6.1		EXOR-ciser	40	2	38.4
Cromemco	SCC-W	8/16	Z-80A	4	0.512	1	0/8	•	S-100	24	1	76.8
Data General	MBC/1	16/15	mN601	8.3	32	2	0/4		P	32	1	9.6
	MicroNova	16/15	mN601	8.3	32	2/4	0.5/4		P		1	16.6M
Digital Eqpt. Corp.	KD11-F	16/15	LSI-11	2.6	32	4/4	0	•	P		1	
Digitek	DK-6800	8/16	6800	0.1/1	48	1	1/4		P	16	1	19.2
Dynabyte	BC1-1	8/16	Z-80	2.5	64	8/16	0/12	•	P	88	4	9.6
Fairchild	OCM-1	8/16	3850	2	64	1	1/10		P	64	1	110b

Interrupt Provisions	Multiprocessing Capability	Counter-Timers: No. of Timers/Bits per timer	Software: A Assembler; AP Applications Package; DB Debugging Aids, Monitor, Etc.; H High-level Language(s); OS Operating System	Supply Voltages (U=unregulated inputs; on-card regulators)	Board Size (in.)	Comments
•		2/14	DB	+5	3.9 X 6.3	256 bytes CMOS RAM with battery backup. Eurocard version, SMP-80
			A, DB, H, OS	+5	10.5 X 12	Maximum addressable ROM 6K. EPROM programming for S6834. Disassembler available. EVK-100, EVK-200 also available
			N.A.	±5, ±12	6 X 8	8-bit A/D converter. I/O port enables only. Power-on reset
			A, DB, H, AP, OS	±5, ±12	8.5 X 14	TTL serial interface, composite video output, cassette I/O port, ASCII keyboard input, audio output for loudspeaker, two joystick inputs. Parallel I/O is by bus accessory cards.
•			H	+5	4.5 X 6	Software includes executive and emulation programs in PROMs; also communications software. Optional: Z-80 CPU, multiprocessing capability, board size
			A, H, DB	+5	6 X 9.75	Two boards differ only in I/O capabilities
•	•		A, DB, H, OS	+5	7.5 X 16.9	Power fail shutdown and auto restart. Real-time clock. Pascal available. Multiply and divide instructions standard
			A, AP, DB, H, OS	±5, ±12	16 X 8	User's program library available for both types
•	•		A, AP, DB, H, OS	±5, ±12	10 X 7	Serial ports TTL, to 9.6 Kbaud asynch., 50 Kbaud synch. Three interrupt inputs
•		3/16	DB, AP	±5, ±12	6 X 9.75	Compatible with Motorola EXORciser boards
		5/16	A, DB, H, OS	+8, ±18 U	5 X 10	Mates with all S-100 compatible computers
			A, AP, SB, H, OS	+5, +15, -5	10.4 X 7.5	Software includes most Nova and Eclipse programs
			A, AP, SB, H, OS	+5, +15, -5	9.5 X 7.5	I/O is bit serial at 16.6 MHz. Software includes most Nova and Eclipse programs
			A, AP, DB, H, OS	+5, +12	10.5 X 8.9	Jumper-selected restart mode. Double-precision fixed and floating-point arithmetic and multiply/divide options. Software-compatible with PDP-11
			OS, H, A, DB, AP	+5, +12	8 X 8.65	Many additional support cards are available
•			A, H	±5, ±12, +28	14.8 X 12.4	For control applications. 8 relay outputs, 16 LEDs, on-board EPROM programmer. Cassette port and video output
•		2/16	DB	+5	7.5 X 10.5	Includes TTY operating system and programmable baud rates

(*Continued*)

Manufacturer	Model	Word Size in Bits (data/address)	CPU Type	Clock Freq. (MHz) Min./Max.	Memory Kbytes (K-1024) Total Addressable	Amount of RAM on Card (b=bytes)	Amount of ROM on Card[a]	DMA Capability	Bus Type (P-proprietary, blank-no bus)	Parallel I/O Lines	Serial I/O Number of Ports	Serial I/O Baud Rate (max.) (k Baud)
Fairchild (*Cont.*)	GLOW-16	16/16	9440	8/12	512	64	8	•	Nova	32	2	9.6
	SPARK-16	16/16	9440	8/12	64	32	4/4	•	Nova	0	1	9.6
Henize Interactive Control	M-5002	8/16	MC6802	0.9216/2	See comments		1/4	•	P	20	1	9.6
Heurikon	MLZ-80	8/16	Z-80	2 or 4	64	4	0/8	•	Multibus	32	2	19.2
Hewlett-Packard	2108K	16/16	Custom	18.5	2M	See comments		•	21Mx	16	0	
	2109K			28.5								
Imsai	MPU-B	8/16	8085	3	64/1M	0.25	2/4	•	S-100	8	2	56
Intel	iSBC 80/04	8/16	8085 A	1.955	See comments	0.25	2/4		none	22	1	4.8
	iSBC 80/05	8/16	8085 A	1.966	64	0.5	2/4	•	Multibus	22	1	4.8
	iSBC 80/10A	8/16	8080 A	2.048	64	1	0/8	•	Multibus	48	1	38.4
	iSBC 80/20	8/16	8080 A	2.15	64	2	0/8	•	Multibus	48	1	38.4
	iSBC 80/30	8/16	8085 A	2.76	64	16/16[1]	0/9[1]	•	Multibus	24/48[1]	1	38.4
	iSBC 86/12	16/16	8086	5	1000	32	0/16	•	Multibus	24	1	38.4
MicroDaSys	MD-690a	8/16	MC 6802	1	56	1152b	1/10		S-100	16	1	24
Monolithic Systems	MSC-8001	8/16	Z-80A	4	64	4/8	0/16	•	Multibus	48	1	
	MSC-8004	8/16	Z-80 A	2/4	64	4/32	0/16		Multibus	56	1	56

Interrupt Provisions	Multiprocessing Capability	Counter-Timers: No. of Timers/Bits per timer	Software: A Assembler; AP Applications Package; DB Debugging Aids, Monitor, Etc.; H High-level Language(s); OS Operating System	Supply Voltages (U=unregulated inputs; on-card regulators)	Board Size (in.)	Comments
•			A, H, DB	+5, +15	15 X 15	Compatible with the Data General Nova series, and includes 16 switch-selectable baud rates
•			DB	+5	8 X 10	Smaller version of GLOW-16
•	•		A, AP, OS	+5, ±12	9.75 X 5.98	Maximum addressable memory 61,440 bytes. RAM on card: 128 bytes provided, 1152 maximum. Power fail detection and restart. Optional battery backup for RAM. Accessory boards are Motorola-compatible
•	•	4/16	A, HL, OS	+5, ±12	6.75 X 12	On-card floppy-disc controller. Power-on jump to selectable start address. Bus watchdog timer. 16 Mbyte address capability possible. SDLC compatible, with hardware CRC. System bus not required for on-card operations
•		•	A, AP, DB, H, OS	+5 -2	18.13 X 13	Maximum addressable RAM 1 Mword. User-microcodable 4 or 16 Kwords of ROM on card is for microcode. ROM words are 24 bits. Can address 16 Kwds of microcode off-card. DMA transfer to 2 Mwds/sec. 16 words of RAM on card. Member of 21MX family
•	•	3/16	A, DB, H, OS	+8, ±18 U	5.25 X 10	One megabyte of RAM addressable thru auxiliary controller. Power-on jump
•		1/14	A, DB, H, OS	+5	6.75 X 7.85	Total addressable RAM 256 bytes; total addressable ROM 4 Kbytes. For stand-alone applications. Serial I/O via CPU's, SID, SOD lines
•	•	1/14	A, DB, H, OS	+5	6.75 X 12	Same features as 04 but plugs into Multibus
•	•		A, DB, H, OS	±5, ±12	6.75 X 12	Second-sourced by National Semiconductor and other companies
•	•		A, DB, H, OS	+5, +12	6.75 X 12	iSBC 80/20-4: 4 Kbytes of RAM on card. Up to 16 CPUs on bus
•	•	2/16	A, AP, DB H, OS	±5, ±12	6.75 X 12	Board has dual-port memory and socket for an 8041/8741 universal peripheral interface
•	•	2/16	A, AP, DB, H, OS	±5, ±12	6.75 X 12	Has dual-port memory, 9 levels of vectored interrupt and a 1-Mbyte addressing range
	•		DB	+8, ±16 U	5.38 X 10	MC6809 CPU also available. MIKBUG-compatible monitor designed to interface with most fast memory-mapped video and graphics boards
•	•	2/16	DB, OS	+5	6.75 X 12	Z-80 based. Multibus-compatible microcomputer
	•	3/16	A, DB, OS	±5, ±12	6.75 X 12	Board has math chip for floating point processing, one time does band-rate generation for serial I/O

(*Continued*)

					Memory Kbytes (K-1024)						Serial I/O	
Manufacturer	*Model*	*Word Size in Bits (data/address)*	*CPU Type*	*Clock Freq. (MHz) Min./Max.*	*Total Addressable*	*Amount of RAM on Card (b=bytes)*	*Amount of ROM on Card*[a]	*DMA Capability*	*Bus Type (P-proprietary, blank-no bus)*	*Parallel I/O Lines*	*Number of Ports*	*Baud Rate (max.) (k Baud)*
Mostek	OEM-80	8/16	MK 3880	0.005/2.5	64	4/64	0/25	•	P	40	1	9.6
	MD-SBC-1	8/16	Z-80A	0.5/4	64	0.2/2	0/8		STD	40	0	
Motorola	M68MM01	8/16	MC 6800	1	41	1	0/4	•	EXORciser	60		
	M68MM01A	8/16	MC 6800	1	64	1	0/8	•	EXORciser	40	1	9.6
	M68MM01B	8/16	MC 6802	1	42	128b	0/8	•	EXORciser	26	1	9.6
	M68MM01B1A	8/16	MC 6802	1	64	128b 384b	0/8	•	EXORciser	26	1	9.6
	M68MM01D	8/16	6800	1	56	0/10	0/10		EXORciser	8	1	28.8
	M68MM19	8/16	6809	2	51	0/8	0/12		EXORciser	16	1	28.8
Mupro	MBC-80/10X	8/16	8080 A	2	64	1/15	0/23	•	Multibus	48	1	38.4
	MBC-80/20X	8/16	8080 A	2.15	64	1/15	0/23	•	Multibus	48	1	38.4
	MBC-80/40X	8/16	8080A	2.15	64	1/15	0/23	•	Multibus	48	1	38.4
National Semiconductor	IMP-16C,L	16/16	IMP-16	5.7143	64	1	0/1	•	P	16	1	
	ISP-8C100N	8/16	ISP-8A/600	4	64	0.25	0/0.5		P	5	1	9.6
	BLC-80/05	8/16	8085 A	1.966	64	5126	2/8		Multibus	22	1	CPU limited
	BLC-80/07	8/16	8080 A	2.048	64	5126	0/4		Multibus	24		
	BLC-80/10 11, 12, 14	8/16	8080A	2.048	64	See comments	•		Multibus	48	1	38.4
	BLC-80/204	8/16	8080 A-2	2.1504	64	4	0/8		Multibus	48	1	36.4

Interrupt Provisions	Multiprocessing Capability	Counter-Timers: No. of Timers/Bits per timer	Software: A Assembler, AP Applications Package, DB Debugging Aids, Monitor, Etc., H High-level Language(s), OS Operating System	Supply Voltages (U=unregulated inputs; on-card regulators)	Board Size (in.)	Comments
	•	4/16	A, DB, H, OS	+5, ±12	12 X 8.5	Same as Z-80 CPU, ROM and RAM address mapping. OEM-80: No ROM supplied. Available is a complete ROM-based prototype package. European card: 233 X 250 mm
		See comments	A, DB	+5	4.5 X 6.5	Counter/timers in lieu of I/O on some boards
	•	•	A, DB, H	+5, -12	9.75 X 5.98	Has 3 PIAs, 120 I/O lines total. Suitable for control applications. Power-on reset. EXbug can be used
•	•		A, DB, H	+5, ±12	9.75 X 5.98	Has 2 PIAs. Power-on reset. EXbug can be used
•	•	3/16	A, DB, H	+5	9.75 X 5.98	Battery can back up lower 32 bytes of RAM in low-power mode. Power on reset. EXbug can be used
•	•	2/16	A, DB, H	+5, ±12	9.75 X 5.98	Battery can back up lower 32 bytes of RAM in low-power mode. Power-on reset. EXbug can be used. Built-in dynamic RAM refresh circuits. Power-on refresh circuits. Power-on reset. EXbug can be used.
			A, AP, DB, H, OS	+5, ±12	6.15 X 9.75	Software support through EXORciser development library
			DB, H	+5, ±12	6.15 X 9.75	RAM includes 2 Kbytes with battery backup
•			DB	±5, ±12	6.75 X 12	User software libraries and time-shared cross-software available
•	•		A, DB, OS	±5, ±12	6.75 X 12	Extremely large PROM space for applications programs
•	•	3/16	A, DB, OS	±5, ±12	6.75 X 12	Hardware arithmetic processor, calendar, and peripheral processor included
•	•		OS	+5, -12	11 X 8.5	Multiprocessing, and 4 DMA ports (to 1 Mwds) on -16L only. Battery backup line. External clock possible. ROMs extend instruction set. Hi-speed (97) Kwds/sec) block transfer instructions
•	•		DB, HL	+5	4.38 X 4.86	SC/MP II. Delay instruction (132 ms maximum). Suffix NE: Eurocard
•		1/16	A, AP, DB, H, OS	+5	6.75 X 12	Equivalent to Intel's iSBC-80/05 CPU board
•			A, AP, DB. H, OS	±5, ±12	6.75 X 12	Reduced memory and I/O version of the BLC-80/10 board
•	•		DB	±5, ±12	6.75 X 12	BLC-80/10 is Intel iSBC-80/10 equivalent BLC-80/11 is Intel-10A equivalent BLC-80/14 is same as BLC-80/12, except has 4K of RAM
•	•	2/16	A, AP, DB, H, OS	+5, ±12	6.75 X 12	Equivalent to Intel's iSBC-80/20-4 CPU board

(Continued)

					Memory Kbytes (K-1024)						Serial I/O	
Manufacturer	Model	Word Size in Bits (data/address)	CPU Type	Clock Freq. (MHz) Min./Max.	Total Addressable	Amount of RAM on Card (b=bytes)	Amount of ROM on Card[a]	DMA Capability	Bus Type (P-proprietary, blank-no bus)	Parallel I/O Lines	Number of Ports	Baud Rate (max.) (k Baud)
Omnibyte	OB 8001	8/16	6800	1	64	1152b	0/4		P		1	19.2
Pragmatic Designs	CPU-1	8/16	8085 A	3	48	0.5	0/4		P	22	2	CPU limited
Pro-log	PLS-800	8/16		1/3	8	1/2	0/8		P	40	1	9.6
	7801, 2, 3	8/16	8085 6800 Z-80	1/3.125	64	1/4	0/8	•	STD	0	1	CPU limited
Process Computer Systems	PCS 1806	8/16	8080A	2	64	1	0/7	•	Flexibus II	16	1	9.6
	PCS 1810	8/16	8080A	2	64	1	0/3	•	Flexibus II	32	1	9.6
	PCS 1880	8/16	Z-80	4	64	1	0/6	•	Flexibus II	8	1	9.6
Processor Technology	Sol-PC	8/16	8080 A	2.05	64	2	2	•	S-100	16	1	9.6
Quay	90/94 MPS	8/16	Z-80A	2.5/4	64	5/65	1/14	•	P	64	1	9.6
	90F/MPS	8/16	Z-80A	2.5/4	79	64	1/14	•	P	36	1	9.6
RCI/Data	CMC 68/04	8/16	680Z	1	64	38b	0/4		P	16	1	
	CMC 68/15	8/16	6800	1/2	64	2	0/8	•	P	32	1	
Relational Memory	RSBC-Z80/24	8/16	Z-80A	3.9	1048	20	2/24	•	Multibus	16	2	550
Space Byte	8085 CPU	8/16	8085	3	64	0.25	0/6	•	S-100	32	2	38.4
Synertek	CP 110	8/16	6502	1	64	1	1/5	•	P	28	3	9.6
Texas Instruments	TM 990/180M	8/14	TMS9980	2.5/3	16	0.5/1	2/4	•	P	24	1	38.4
	TM 990/100M	16/15	TMS9900	3	32	0.5/1	2/4	•	P	16	1	38.4

Interrupt Provisions	Multiprocessing Capability	Counter-Timers: No. of Timers/Bits per timer	Software: A Assembler, AP Applications Package, DB Debugging Aids, Monitor, Etc., H High-level Language(s), OS Operating System	Supply Voltages (U=unregulated inputs; on-card regulators)	Board Size (in.)	Comments
			A, DB, H, OS	±5, ±12	4.5 X 6.5	Parallel I/O: 16 programmable lines
•		1/14		+5	6.5 X 4	Serial port built from 8085 SID/SOD lines, breadboard area
			A, AP, DB, H	±5, +12	4.5 X 6.5	Five models available—two with 8080A CPU, one with 6800 and one with Z80 CPU. Serial I/O available only on 8085-based card
•	•			+5	6.5 X 4.5	Three CPUs are functionally compatible; baud rate depends on microprocessor
•		5/8	A, DB, H, OS	±5, ±12	10.5 X 8.5	Power fail interrupt
•		5/8	A, DB, H, OS	±5, +5 to +30	10.5 X 8.5	Power fail interrupt
•		1/16	A, DB, H, OS	±5, ±12	10.5 X 8.5	Optional AMD 9511 hardware math chip with fixed, float, and conversions; logs, trig, etc.
•			A, DB, H, OS	±5, ±12	16 X 10	Six configurations. All have 1024-character video out. 2K OS in ROM, audio cassette interface, keyboard interface
	•	4/16	DB, H	±5, +12, +28	16.18 X 7.88	CPU: Z-80 or Z-80A. PROM programmer. Single-step circuits. Hardware breakpoints. Total board memory to 72 K. Backplane-independent
•		1	H, DB, OS	±5, +12, +28	16.175 X 7.875	Paging on upper 16 Kbyte of memory provides a total of 79 Kbytes
		1	A, AP	+5	7.6 X 6.3	Firmware package available for use with HP5004A signature analyzer
•			A, DB	+5	7.6 X 11	Software can be developed with the Motorola EXORciser
•	•		A, DB	±5, ±12	6.75 X 12	Up to 22 Kbytes of ROM can be replaced by static RAM, in-circuit emulator available
•	•	1/14	A, AP, DB, H, OS	+8, ±16U	5 X 10	Parallel I/O port is intended for floppy disc interface
•		1/16	A, DB, H	+5, +12, -10	4.25 X 7	Also called Super Jolt. 64 bytes of interrupt vector RAM on board. Microcomputer with keyboard, called VIM-1, also available
•		2/16	A, DB, H	+5, ±12	11 X 7.5	Serial data can be thru differential line driver/receiver. Prototyping area on board
•		2/16	A, DB, H	+5, ±12	11 X 7.5	Multiply, divide, bit I/O instructions. Prototyping area on board. Another board also available. TM990/101M

(*Continued*)

					Memory Kbytes (K-1024)						Serial I/O	
Manufacturer	*Model*	*Word Size in Bits (data/address)*	*CPU Type*	*Clock Freq. (MHz) Min./Max.*	*Total Addressable*	*Amount of RAM on Card (b=bytes)*	*Amount of ROM on Card*[a]	*DMA Capability*	*Bus Type (P-proprietary, blank-no bus)*	*Parallel I/O Lines*	*Number of Ports*	*Baud Rate (max.) (k Baud)*
Western Digital	Microengine	16/16	WD-9000	3	64	64	0	•	P	16	2	19.2
Wintek	WINCE CMM	8/16	6800	0.1/1	64	0.5	.5/4	•	P	32	1	9.6
Xitan	The General	8/16	Z-80A	2/4	1024	128	1/24	•	S-100	3 ports	0	
Zilog	Z80-MCB	8/16	Z-80	2.47	64	4/16	0/8	•	P	16	1	38.4

[a]*Provided/max. possible.*

Interrupt Provisions	Multiprocessing Capability	Counter-Timers: No. of Timers/Bits per timer	Software: A Assembler, AP Applications Package, DB Debugging Aids, Monitor, Etc., H High-level Language(s), OS Operating System	Supply Voltages (U=unregulated inputs; on-card regulators)	Board Size (in.)	Comments
			H	±5, ±12	8 X 16	Executes programs directly from Pascal P-code without assembly or compiling
	•		A, DB, H	+5, ±12	4.5 X 6.5	44-pin/0.156 in. connectors. Some unusual accessory cards
•	•	3/16	A, DB, H, OS	±5, ±12	12 X 16	For use in smart terminal; can execute four tasks concurrently
•	•	4/16	A, AP, DB, H, OS	+5	7.7 X 7.5	Board has a 126-pin interface bus

bibliography

Computer Architecture

Bell, C. G., and Newell, A., *Computer Structures*, McGraw-Hill, New York, 1970.

Bell, C. G., Mudge, J. C., and McNamara, J. E., *Computer Engineering*, Digital, Bedford, Mass., 1978.

Chu, Y. (Ed.), *High-Level Language Computer Architecture*, Academic, New York, 1975.

Foster, C. C., *Computer Architecture*, Van Nostrand Reinhold, New York, 1970.

Foster, C. C., *Content Addressable Parallel Processors*, Van Nostrand Reinhold, New York, 1976.

Lorin, H., *Parallelism in Hardware and Software: Real and Apparent Concurrency*, Prentice-Hall, Englewood Cliffs, N.J., 1972.

Organick, E., *Computer System Organization: The B5700/B6700 Series*, Academic, New York, 1973.

Stone, H. S., *Introduction to Computer Organization and Data Structures*, McGraw-Hill, New York, 1972.

Stone, H. S., (Ed.), *Introduction to Computer Architecture*, Science Research Associates, Palo Alto, Calif., 1975.

Thurber, K. J., *Large Scale Computer Architecture*, Hayden, Rochelle Park, N.J., 1976.

Digital Electronics

Blakeslee, T. R., *Digital Design with Standard MSI and LSI*, (2nd ed.) Wiley-Interscience, New York, 1979.

Kohonen, T., *Digital Circuits and Devices*, Prentice-Hall, Englewood Cliffs, N.J., 1972.

Lee, S. C., *Digital Circuits and Logical Design*, Prentice-Hall, Englewood Cliffs, N.J., 1976.

Taub, H., and Schilling D., *Digital Integrated Electronics*, McGraw-Hill, New York, 1977.

Microprocessors

Barna, A., and Porat, D. L., *Introduction to Microcomputers and Microprocessors*, Wiley-Interscience, New York, 1976.

Garland H., *Introduction to Microprocessor System Design*, McGraw-Hill, New York, 1979.

Givone, D. D., and Roesser, R. P., *Microprocessors/Microcomputers—An Introduction*, McGraw-Hill, New York, 1980.

Hilbrun, J. L., and Julich, P. M., *Microcomputers/Microprocessors: Hardware, Software and Applications*, Prentice-Hall, Englewood Cliffs, N.J., 1976.

Klingman, E. E., *Microprocessor System Design*, Prentice-Hall, Englewood Cliffs, N.J., 1977.

McGlynn, D. R., *Microprocessors: Technology, Architecture, and Applications*, Wiley-Interscience, New York, 1976.

Osborne, A., *An Introduction to Microcomputers*, Osborne, Berkeley, 1976.

Peatman, J., *Microcomputer-Based Design*, McGraw-Hill, New York, 1977.

Soucek, B., *Microprocessors and Microcomputers*, Wiley-Interscience, New York, 1976.

Specific Microprocessors

MCS-86 Product Description, Intel Corporation, Santa Clara, Calif., 1979.

SDK-86, MCS-86 System Design Kit User's Guide, Intel Corporation, Santa Clara, Calif., 1978.

MCS-86 User's Manual, Intel Corporation, Santa Clara, Calif., 1978.

ASM86 Language Reference Manual, Intel Corporation, Santa Clara, Calif., 1978.

Z8001, Z8002 Product Description, Zilog Inc., Cupertino, Calif., 1979.

Z8000 Instruction Set, Zilog Inc., Cupertino, Calif., 1979.

Periodicals

Computer Design
Monthly, controlled circulation
Computer Design Publishing Corp.
11 Goldsmith Street
Littleton, MA 01460

EDN
Bi-weekly, controlled circulation
Cahners Publishing Co.
221 Columbus Ave.
Boston, MA 02116

Electronic Design
Bi-weekly, controlled circulation
Hayden Publishing Co., Inc.
50 Essex Street
Rochelle Park, NJ 07662

Electronics
Bi-weekly, $17 per year
McGraw-Hill, Inc.
1221 Ave. of the Americas
New York, N.Y. 10020

Microelectronics Journal
Quarterly, $14 back issue
Mackintosh Publications Ltd.
Napier Road, Luton
Bedfordshire, England LU1 1RG

Microprocessors and Microsystems
10 issues per year, £15 year for individuals
IPC Science and Technology Press Ltd.
P.O. Box 63, Westbury House
Guildford, Surrey GU2 5BH, England

glossary*

Absolute Program. A program that incorporates absolute addresses and therefore must occupy the same area of memory each time it is executed.

Access Control. The tasks imposed on a network or any of its components—performed by hardware, software, and administrative controls—to control usage of the system. Included are monitoring of system operation, insuring of data integrity, user identification, recording system access and changes, and methods for granting users access.

Access Method. A data management technique for transferring data between memory and an input/output device (e.g., direct, indexed, keyed, queued, or sequential access). Also called **Communications.**

Access Time. Time between the instant that an address is sent to a memory and the instant that data returns. Since the access time to different locations (addresses) of the memory may be different, the access time specified in a memory device is the path that takes the longest time.

Accumulator. Register and related circuitry that holds one operand for arithmetic and logical operations.

Address. A number used by the CPU to specify a location in memory.

*Reprinted with permission from Daniel R. McGlynn, *Personal Computing* (Wiley, New York, 1979).

Additional Hardware. Microprocessor chips differ in number of additional integrated circuits required to implement a functioning computer. Generally, timing, input/output control, buffering, and interrupt control require external components.

Addressing Modes. Also called **Memory Addressing Modes.**

Allocation. Assigning a resource (e.g., allocating an area of main storage to a task or allocating an input/output device to a job).

ALU (Arithmetic-Logic Unit). That part of a CPU that executes adds, subtracts, shifts, AND's, OR's, and so forth.

Architecture. Organizational structure of a computing system, mainly referring to the CPU or microprocessor.

Assembler. Software that converts an assembly-language program into machine language. The assembler assigns locations in storage to successive instructions and replaces symbolic addresses by machine language equivalents. If the assembler runs on a computer other than that for which it creates the machine language, it is a **Cross-Assembler.**

Assembly Language. An English-like programming language that saves the programmer the trouble of remembering the bit patterns in each instruction; also relieves him of the necessity to keep track of locations

of data and instructions in his program. The assembler operates on a "one-for-one" basis in that each phrase of the language translates directly into a specific machine-language word, as contrasted with **High Level Language.**

Assembly Listing. A printed listing made by the assembler to document an assembly. It shows, line for line, how the assembler interpreted the assembly language program.

Asynchronous Operation. Circuit operation without reliance upon a common timing source. Each circuit operation is terminated (and next operation initiated) by a return signal from the destination denoting completion of an operation. (Contrast with **Synchronous Operation.**)

Asynchronous Transmission. Transmission in which time intervals between transmitted characters may be of unequal length. Transmission is controlled by start and stop elements at the beginning and end of each character. Also called **Start-Stop Transmission.**

ASCII. American Standard Code for Information Interchange. This is a 7-bit-plus-parity code established by the American National Standards Institute (formerly American Standards Association) to achieve compatibility among data services. Also called **USASCII.**

Amplitude Modulation (AM). A method of transmission whereby the amplitude of the carrier wave is modified in accordance with the amplitude of the signal wave.

Availability. A system attribute that means that the system is operable and available to the user even though certain system components may have failed.

Background Processing. The automatic execution of a low-priority computer program when higher priority programs are not using the system resources.

Bandwidth. The range of frequencies assigned to a channel or system. The difference expressed in Hertz between the highest and lowest frequencies of a band.

Baud. A communications measure of serial data transmission rate; loosely, bits per second, but includes character-framing START and STOP bits.

Benchmark Program. A sample program used to evaluate and compare computers. In general, two computers will not use the same number of instructions, memory words, or cycles to solve the same problem.

Bit. An abbreviation of "binary digit." (Single characters in a binary number.)

Bit Transfer Rate. The number of bits transferred per unit time, usually expressed in Bits Per Second (BPS).

BPS. Bits per second.

Block. A group of digits transmitted as a unit, over which a coding procedure is usually applied for synchronization or error control purposes. See also **Packet.**

Bootstrap (Bootstrap Loader). Technique or device for loading first instructions (usually only a few words) of a routine into memory; then using these instructions to bring in the rest of the routine. The bootstrap loader is usually entered manually or by pressing a special console key. See also **Load Facility.**

Branch. See **Jump.**

Branch Instruction. A decision-making instruction that, on appropriate condition, forces a new address into the program counter. The conditions may be zero result, overflow on add, an external flag raised, etc. One of two alternate program segments in the memory is chosen, depending on the results obtained.

Breakpoint. A location specified by the user at which program execution (real or simulated) is to terminate. Used to aid in locating program errors.

Bus. A group of wires that allow memory, CPU, and input/output devices to exchange words.

Byte. A sequence of n bits operated upon as a unit is called an n-bit byte. The most frequent byte size is 8 bits.

Buffer. A storage device used to compensate for a difference in the rate of data flow when transmitting data from one device to another.

Call Routine. Also called **Subroutine.**

Cathode-Ray Tube (CRT). A television-like picture tube used in visual display terminals.

Channel. That part of a communications system that connects a message source to a message sink. See also **Information (Transfer) Channel.**

Channel Capacity. A term that expresses the maximum bit rate that can be handled by the channel.

Circuit. In communications, the complete electrical path providing one or two-way communication between two points comprising associated go and return channels. Compare **Channel.**

Clock. A device that sends out timing pulses to synchronize the actions of the computer.

Code. (1) A set of unambiguous rules specifying the way in which data may be represented (e.g., the set of correspondences in the Standard Code for Information Interchange). (2) In data communications, a system of rules and conventions according to which the signals representing data can be formed, transmitted, received, and processed. (3) In data processing, to represent data or a computer program in a symbolic form that can be accepted by a data processor.

Compiler. Software to convert a program in a high-level language such as FORTRAN into an assembly language or machine language program.

Computer Network. An interconnection of assemblies of computer systems, terminals, and communications facilities.

Connect Time. A measure of system usage by a user,

usually the time interval during which the user terminal was on line during a session. See also **CPU Time.**

Control Character. (1) A character whose occurrence in a particular context initiates, modifies, or stops a control function. (2) In the ASCII code, any of the 32 characters in the first two columns of the standard code table. See also **Communications Control Character.**

Conversational. Pertaining to a mode of processing that involves step-by-step interaction between the user at a terminal by means of keyboard and display and a computer. See also **Interactive.**

CPU (Central Processing Unit). That part of a computer system that controls the interpretation and execution of instructions. In general, the CPU contains the following elements: Arithmetic-Logic Unit (ALU), Timing and Control, Accumulator, Scratch-pad memory, Program counter and address stack, Instruction register and decode, Parallel data and input/output bus, Memory and input/output control.

CPU Time. Central Processing Unit Time. A measure of system usage by a user, based on the total amount of computer processing time used. See also **Connect Time.**

Cross Assembler. A symbolic language translator that runs on one type of computer to produce machine code for another type of computer. See **Assembler.**

Cycle Stealing. A memory cycle stolen from the normal CPU operation for a DMA operation. See **DMA.**

Cycle Time. Time interval at which any set of operations is repeated regularly in the same sequence.

Data Access Arrangement (DAA). Data communication equipment furnished by a common carrier, permitting attachment of privately owned data terminal and data communication equipment to the common carrier network.

Data Acquisition. The retrieval of data from remote sites initiated by a central computer system (e.g., retrieving data during off-hours processing from a previously mounted magnetic tape at an unattended terminal or taking periodic readings from an unattended real-time station).

Data Base. (1) The entire collection of information available to a computer system. (2) A structured collection of information as an entity or collection of related files treated as an entity.

Data Base Management System (DBMS). Facilities for creating and maintaining a large, organized, structured collection of data and providing means for producing various types of predetermined reports based upon interrogation of the data base.

Data Collection. The act of bringing data from one or more points to a central point.

Data Communication. The interchange of data messages from one point to another over communications channels. See also **Data Transmission.**

Data Communication Equipment. The equipment that provides the functions required to establish, maintain, and terminate a connection, the signal conversion, and coding required for communication between data terminal equipment and data circuit. The data communication equipment may or may not be an integral part of a computer (e.g., a modem). See also **Terminal Installation, Data Link.**

Data Compression. A technique whereby a repetitive string of data (usually on a byte basis) is transmitted as a count plus a string value.

Data Concentration. Collection of data at an intermediate point from several low- and medium-speed lines for retransmission across high-speed lines.

Data Integrity. A performance measure based on the rate of undetected errors.

Data Link. An assembly of terminal installations and the interconnecting circuits operating according to a particular method that permits information to be exchanged between terminal installations. NOTE: The method of operation is defined by particular transmission codes, transmission modes, direction, and control.

Data Set. (1) A modem. (2) A collection of data records, with a logical relation of one to another. See also **Data-Phone Set, Modem.**

Data Terminal Equipment (DTE). (1) The equipment comprising the data source, the data sink, or both. (2) Equipment usually comprising the following functional units: control logic, buffer store, and one or more input or output devices or computers. It may also contain error control, synchronization, and station identification capability. See also **Data Communication Equipment, Data Link, Terminal Installation.**

Data Transmission. The sending of data from one place for reception elsewhere. Compare **Data Communication.**

Data Pointer. A register holding the memory address of the data (operand) to be used by an instruction. Thus the register "points" to the memory location of the data.

Data Register. Any register that holds data.

Debug. To eliminate programming mistakes, including omissions, from a program.

Debug Programs. Debug programs help the programmer to find errors in his programs while they are running on the computer, and allow him to replace or patch instructions into (or out of) his program.

Diagnostic Programs. These programs check the various hardware parts of a system for proper operation; CPU diagnostics check the CPU, memory diagnostics check the memory, and so forth.

Dial-Up Line. A communications circuit that is established by a switched circuit connection.

Direct Addressing. The address of an instruction or

operand is completely specified in an instruction without reference to a base register or index register.

DMA (Direct Memory Access). A mechanism that allows an input/output device to take control of the CPU for one or more memory cycles, in order to write to or read from memory. The order of executing the program steps (instructions) remains unchanged.

EBCDIC. Extended Binary Coded-Decimal Interchange Code. An 8-bit character code used primarily in IBM equipment. The code provides for 256 different bit patterns.

Editor. As an aid in preparing source programs, certain programs have been developed that manipulate text material. These programs, called editors, text editors, or paper tape editors make it possible to compose assembly-language programs on-line, or on a stand-alone system.

Effectiveness. A system attribute that measures the level or degree to which the system is attaining predetermined goals.

Efficiency. A system attribute that measures whether the system is making optimum use of the system resources.

Error Control. An arrangement or technique for the detection of errors, and often correction of errors as well, by operations on the received data or retransmission from the source. See also **ARQ, FEC.**

Error Correction Code. A code in which the data signals conform to specific code rules so that departures in these rules at the receiving station can be automatically detected and corrected (e.g., cyclic block codes, convolutional codes).

Error Rate. The ratio of the number of bits or other message units incorrectly received to the total number of bits of message units transmitted.

Extensibility. A system attribute that permits the addition of new, user-defined functions to the system.

Execute. The process of interpreting an instruction and performing the indicated operation(s).

FEC (Forward Error Correction). A technique of error control in which errors in a transmitted message are corrected through the use of redundant information at the receiving station.

Fetch. A process of addressing the memory and reading into the CPU the information word, or byte, stored at the addressed location. Most often, fetch refers to the reading out of an instruction from the memory.

Firmware. Software that is implemented in ROMs.

Fixed-Instruction Computer (Stored-Instruction Computer). The instruction set of a computer is fixed by the manufacturer. The users will design application programs using this instruction set (in contrast to the **Microprogrammable Computer** for which the users must design their own instruction set and thus customize the computer for their needs.)

Fixed Memory. Also called **ROM.**

Flag Lines. Inputs to a microprocessor controlled by input/output devices and tested by branch instructions.

Fortran. A high-level programming language generally for scientific use, expressed in algebraic notation. Short for "Formula Translator."

Guard. A mechanism to terminate program execution (real or simulated) upon access to data at a specified memory location. Used in debugging.

Handshaking. The exchange of predetermined signals when a connection is established between two stations for the purpose of control and synchronization.

Hardware. Physical equipment forming a computer system.

Hashing. A technique of calculating the address of data in memory from its value.

Hexadecimal. Number system using 0, 1, ..., A, B, C, D, E, F to represent all the possible values of a 4-bit digit. The decimal equivalent is 0–15. Two hexadecimal digits can be used to specify a byte.

High-Level Language. Programming language that generates machine codes from problem- or function-oriented statements. FORTRAN, COBOL, and BASIC are three commonly used high-level languages. A single functional statement may translate into a series of instructions or subroutines in machine language, in contrast to a low-level (assembly) language in which statements translate on a one-for-one basis.

IC. Integrated circuit.

Interactive. Pertaining to exchange of information and control between a user and a computer process, or between computer processes. See also **Conversational.**

Interface Standard. A standardized method for establishing an interface (e.g., the Electronic Industries Association (EIA) RS-232C, the Department of Defense MIL STD 188B, or the CCITT standard).

Immediate Addressing. The method of addressing an instruction in which the operand is located in the instruction itself or in the memory location immediately following the instruction.

Immediate Data. Data that immediately follow an instruction in memory, and are used as an operand by that instruction.

Indexed Addressing. An addressing mode, in which the address part of an instruction is modified by the contents in an auxiliary (index) register during the execution of that instruction.

Index Register. A register that contains a quantity which may be used to modify memory address.

Indirect Addressing. A means of addressing in which the address of the operand is specified by an auxiliary

register or memory location specified by the instruction rather than by bits in the instruction itself.

Input-Output (I/O). General term for the equipment used to communicate with a computer CPU; or the data involved in that communication.

Instruction. A set of bits that defines a computer operation, and is a basic command understood by the CPU. It may move data, do arithmetic and logic functions, control input/output devices, or make decisions as to which instruction to execute next.

Instruction Cycle. The process of fetching an instruction from memory and executing it.

Instruction Length. The number of words needed to store an instruction. It is one word in most computers, but some will use multiple words to form one instruction. Multiple-word instructions have different instruction execution times depending on the length of the instruction.

Instruction Repertoire. Also called **Instruction Set.**

Instruction Set. The set of general-purpose instructions available with a given computer. In general, different machines have different instruction sets. The number of instructions only partially indicates the quality of an instruction set. Some instructions may only be slightly different from one another; others rarely may be used. Instruction sets should be compared using benchmark programs typical of the application, to determine execution times, and memory requirements.

Instruction Time. The time required to fetch an instruction from memory and then execute it.

Interpreter. A program that fetches and executes "instructions" (**pseudo instructions**) written in a higher level language. The higher-level language program is a **pseudo program.** Contrast with **Compiler.**

Interrupt Request. A signal to the computer that temporarily suspends the normal sequence of a routine and transfers control to a special routine. Operation can be resumed from this point later. Ability to handle interrupts is very useful in communication applications where it allows the microprocessor to service many channels.

Interrupt Mask (Interrupt Enable). A mechanism that allows the program to specify whether or not interrupt requests will be accepted.

Interrupt Service Routine. A routine (program) to properly store away to the stack the present status of the machine in order to respond to an interrupt request; perform the "real work" required by the interrupt; restore the saved status of the machine; and then resume the operation of the interrupted program.

I/O. Input/output.

I/O Control Electronics (I/O Controller). The control electronics required to interface an input/output device to a computer CPU. The powerfulness and usefulness of a CPU is very closely associated with the range of input/output devices that can be connected to it. One cannot usually simply plug them into the CPU. The input/output Control Electronics will do the "matchmaking". The complexity and cost of the Control Electronics are very much determined by both the hardware and software input/output architecture of the CPU.

I/O Interface. Also called **I/O Control Electronics.**

I/O Port. A connection to a CPU that is configured (or programmed) to provide a data path between the CPU and the external devices, such as keyboard, display, reader, etc. An input/output port of a microprocessor may be an input port or an output port, or it may be bidirectional.

Jump. A departure from the normal one-step incrementing of the program counter. By forcing a new value (address) into the program counter the next instruction can be fetched from an arbitrary location (either further ahead or back). For example, a program jump can be used to go from the main program to a subroutine, from a subroutine back to the main program, or from the end of a short routine back to the beginning of the same routine to form a loop. See also **Branch Instruction.** If you reached this point from Branch, you have executed a Jump. Now Return.

Linkage. Also called **Subroutine.**

Load Facility. A hardware facility to allow program loading using DMA. It makes bootstrap unnecessary.

Loader. A program to read a program from an input device into RAM. May be part of a package of utility programs.

Loop. A self-contained series of instructions in which the last instruction can cause repetition of the series until a terminal condition is reached. Branch instructions are used to test the conditions in the loop to determine if the loop should be continued or terminated.

Low-Level Language. Also called **Assembly Language.**

Machine. A term for a computer (of historical origin).

Machine Code. Also called **Machine Language.**

Machine Cycle. The basic CPU cycle. In one machine cycle an address may be sent to memory and one word (data or instruction) read or written, or, in one machine cycle a fetched instruction can be executed.

Machine Language. The numeric form of specifying instructions, ready for loading into memory and execution by the machine. This is the lowest-level language in which to write programs. The value of every bit in every instruction in the program must be specified (e.g., by giving a string of binary, octal, or hexadecimal digits for the contents of each word in memory).

Machine State. Also called **State Code.**

Macro (Macroinstruction). A symbolic source language statement that is expanded by the assembler into one or more machine language instructions, relieving the programmer of having to write out frequently occurring instruction sequences.

Manufacturer's Support. It includes application information, software assistance, components for prototyping, availability of hardware in all configurations from chips to systems, and fast response to requests for engineering assistance.

Mask. To suspend recognition of a signal.

Memory. That part of a computer that holds data and instructions. Each instruction or datum is assigned a unique address that is used by the CPU when fetching or storing the information.

Memory Address Register. The CPU register that holds the address of the memory location being accessed.

Memory Addressing Modes. The method of specifying the memory location of an operand. Common addressing modes are—direct, immediate, relative, indexed, and indirect. These modes are important factors in program efficiency.

Microcomputer. A computer whose CPU is a microprocessor. A microcomputer is an entire system with microprocessor, memory, and input-out controllers.

Microprocessor. Frequently called "computer on a chip." The microprocessor is, in reality, a set of one, or a few, LSI circuits capable of performing the essential functions of a computer CPU.

Microprogrammable Computer. A computer in which the internal CPU control signal sequence for performing instructions is generated from a ROM. By changing the ROM contents, the instruction set can be changed. This contrasts with a **Fixed-Instruction Computer**, in which the instruction set cannot be readily changed.

Mnemonics. Symbolic names or abbreviations for instructions, registers, memory locations, and so forth. A technique for improving the efficiency of the human memory.

Multiple Processing. Configuring two or more processors in a single system, operating out of a common memory. This arrangement permits execution of as many programs as there are processors.

Nesting. Subroutines that are called by subroutines are said to be nested. The nesting level is the number of times nesting can be repeated.

Network. (1) An interconnected or interrelated group of nodes. (2) In connection with a disciplinary or problem-oriented qualifier, the combination of material, documentation, and human resources that are united by design to achieve certain objectives (e.g., a social science network, a science information network). See also **Computer Network.**

Nibble. A sequence of 4 bits operated upon as a unit. Also see **Byte.**

Object Program. Program that is the output of an automatic coding system, such as the assembler. Often the object program is a machine-language program ready for execution.

On-Line System. A system of input/output devices in which the operation of such devices is under the control of the CPU, and in which information reflecting current activity is introduced into the data-processing or controlling system as soon as it occurs.

Op Code (Operation Code). A code that represents specific operations of an instruction.

Operating System. System software controlling the overall operation of a multipurpose computer system, including such tasks as memory allocation, input and output distribution, interrupt processing, and job scheduling.

Overhead Bit. A bit other than an information bit (e.g., check bit, framing bit).

Packet. A group of bits, including data and control elements, that is switched and transmitted as a composite whole. The data and control elements and possibly error control information are arranged in a specified format.

Packet Switching. A data transmission process utilizing addressed packets, whereby a channel is occupied only for the duration of transmission of the packet. NOTE: In certain data communication networks the data may be formatted into a packet or divided and then formatted into a number of packets (either by the data terminal equipment or by equipment within the network) for transmission and multiplexing purposes. See also **Circuit Switching, Message Switching.**

Page. A natural grouping of memory locations by higher-order address bits. In an 8-bit microprocessor, $2^8 = 256$ consecutive bytes often may constitute a page. Then words on the same page only differ in the lower-order 8 address bits.

Paging. The division of data or programs into fixed-size blocks or "pages" that are loaded into main memory only as required.

Parallel Transmission. Method of data transfer in which all bits of a character or byte are transmitted simultaneously either over separate communication lines or on different carrier frequencies on the same communication line.

Parity Check. Addition of noninformation bits to data, making the number of ones in each grouping of bits either always odd for odd parity or always even for even parity. This permits single error detection in each group.

Partition. A subdivision of main storage into fixed or

variable regions or "partitions" that are allocated to a job or system task.

Password. A word or string of characters that is recognizable by automatic means and that permits a user access to protected storage, files, or input or output devices.

PC. Printed circuit.

Peripheral Device. Any device that is not part of the central processing unit (CPU) in a computer system or network.

PLA (Programmable Logic Array). A PLA is an array of logic elements that can be programmed to perform a specific logic function. In this sense, the array of logic elements can be as simple as a gate or as complex as a ROM. The array can be programmed (normally mask programmable) so that a given input combination produces a known output function.

Pointer. Registers in the CPU that contain memory addresses. See **Program Counter** and **Data Pointer.**

Process. (1) A systematic sequence of operations to produce a specified result. (2) A set of related procedures and data undergoing execution and manipulation by one or more computer processing units.

Program. A collection of instructions properly ordered to perform some particular task.

Program Counter. A CPU register that specifies the address of the next instruction to be fetched and executed. Normally it is incremented automatically each time an instruction is fetched.

PROM (Programmable Read-Only Memory). An integrated-circuit memory array that is manufactured with a pattern of either all logical 0's or 1's and has a specific pattern written into it by the user by a special hardware programmer. Some PROMs, called EAROMs, Electrically Alterable Read-Only Memory, can be erased and reprogrammed.

Prototyping Kit. A hardware system used to breadboard a microprocessor-based product. Contains CPU, memory, basic input/output, power supply, switches and lamps, provisions for custom input/output controllers, memory expansion, and often, a utility program in fixed memory (ROM).

Pseudo Instruction. See **Interpreter.**

Pseudo Program. See **Interpreter.**

RAM (Random Access Memory). Any type of memory that has both read and write capability. It is randomly accessible in the sense that the time required to read from or to write into the memory is independent of the location of the memory where data was most recently read from or written into. In contrast, in a **Serial Access Memory**, this time is variable.

Random Access. Pertaining to a storage device where data or blocks of data can be read in any particular order (e.g., disk or tape). In random-access devices you do not have to read from the beginning to find what you want as you do with paper tape and industry-compatible magnetic tape.

Real Time System. A system performing computation during the actual time the related physical process transpires, so that the results of the computation can be used in guiding the process.

Register. A fast-access circuit used to store bits or words in a CPU. Registers play a key role in CPU operations. In most applications, the efficiency of programs is related to the number of registers.

Relative Addressing. The address of the data referred to is the address given in the instruction plus some other number. The "other number" can be the address of the instruction, the address of the first location of the current memory page, or a number stored in a register. Relative addressing permits the machine to relocate a program or a block of data by changing only one number.

Resource. Any means available to network users, such as computational power, programs, data files, storage capacity, or a combination of these.

Resource Allocation. The assignment of a system resource for the use of a program, partition, job, or task.

Resource Sharing. The joint use of resources available on a network by a number of dispersed users.

Return Routine. Also called **Subroutine.**

ROM. Read-Only Memory (Fixed Memory) is any type of memory that cannot be readily rewritten; ROM requires a masking operation during production to permanently record program or data patterns in it. The information is stored on a permanent basis and used repetitively. Such storage is useful for programs or tables of data that remain fixed and is usually randomly accessible.

Routine. Usually refers to a subprogram, that is, the task performed by the routine is less complex. A program may include routines. See also **Program.**

Scratch-Pad Memory. RAM or registers that are used to store temporary intermediate results (data) or memory addressed (pointers).

Serial Memory (Serial Access Memory). Any type of memory in which the time required to read from or write into the memory is dependent on the location in the memory. This type of memory has to wait while nondesired memory locations are accessed. Examples are paper tape, disc, magnetic tape, CCD, and so forth. In a **Random Access Memory,** access time is constant.

Serial Transmission. A method of transmission in which each bit of information is sent sequentially on a single channel rather than simultaneously as in parallel transmission.

Simulators. Software simulators are sometimes used in the debug process to simulate the execution of machine-language programs using another computer (often a time-sharing system). These simulators are especially useful if the actual computer is not available. They may facilitate the debugging by providing access to internal registers of the CPU which are not brought out to external pins in the hardware.

Snapshots. Capture of the entire state of a machine (real or simulated)—memory contents, registers, flags, and so forth.

Software. Computer programs. Often used to denote general-purpose programs provided by the manufacturer, such as assembler, editor, compiler, and so forth.

Source Program. Computer program written in a language designed for ease of expression of a class of problems or procedures, by humans: symbolic or algebraic.

Stack. A sequence of registers and/or memory locations used in LIFO fashion (last-in-first-out). A **Stack Pointer** specifies the last-in entry (or where the next-in entry will go).

Stack Pointer. The counter, or register, used to address a stack in the memory. See also **Stack.**

Stand-Alone System. A microcomputer software development system that runs on a microcomputer without connection to another computer or a time-sharing system. This system includes an assembler, editor, and debugging aids. It may include some of the features of a prototyping kit.

State Code. A coded indication of what state the CPU is in—responding to an interrupt, servicing a DMA request, executing an input/output instruction, and so forth.

Subroutine. A subprogram (group of instructions) reached from more than one place in a **main program.** The process of passing control from the main program to a subroutine is a **subroutine call,** and the mechanism is a **subroutine linkage.** Often data or data addresses are made available by the main program to the subroutine. The process of returning control from subroutine to main program is **subroutine return.** The linkage automatically returns control to the original position in the main program or to another subroutine. See also **Nesting.**

Subroutine Linkage. See **Subroutine.**

Support. See **Manufacturer's Support.**

Swapping. The technique of storing low-priority programs or data in secondary storage and exchanging or "swapping" such data with data in main storage when necessary.

Synchronous Operation. Use of a common timing source (clock) to time circuit or data transfer operations. (Contrast with **Asynchronous Operation.**)

Syntax. Formal structure. The rules governing sentence structure in a language, or statement structure in a language such as assembly language or Fortran.

Teleprocessing. The general concept of data transmission and processing by a computer system and remotely located devices.

Teletype. Trademark of Teletype Corporation. Usually refers to one of their series of teleprinters.

Terminal. An input/output device at which data leaves or enters a computer system (e.g., teletype terminal, CRT terminal, etc.).

Throughput. A measure of system efficiency as a function of the amount of data handled compared with the theoretical maximal data handling capacity.

Turnaround Time. (1) The elapsed time between submission of a job to a computing center and the return of results. (2) In communications, the actual time required to reverse the direction of transmission from sender to receiver or vice versa when using a two-way alternate circuit. Time is required by line propagation effects, modem timing, and computer reaction.

Unbundling. Pricing certain types of software and services separately from the hardware.

Usability. A system attribute that means that the system hardware and software interfaces to the user are simple, convenient, and easy to use.

Utility Program. A program providing basic conveniences, such as capability for loading and saving programs, for observing and changing values in a computer, and for initiating program execution. The utility program eliminates the need for "reinventing the wheel" every time a designer wants to perform a common function.

Virtual Address. A relative location in main or secondary storage that must be converted to a real address before use.

Word. The basic group of bits that is manipulated (read in, stored, added, read out, etc.) by the computer in a single step. Two types of words are used in every computer: data words and instruction words. Data words contain the information to be manipulated. Instruction words cause the computer to execute a particular operation.

Word Length. The number of bits in the computer word. The longer the word length, the greater the precision (number of significant digits). In general, the longer the word length, the richer the instruction set and the more varied the addressing modes.

index

Access time, 183-184
Accuracy-speed product, 234
A/D converter, 230, 233-234
Address, effective, 86
ADIU, 166
AMD:
2901, 148-154, 167-181
2903, 154-158
2909, 168
2910, 168
2911, 168
29116, 64, 66-67
8224, 227
8238, 227
9511, 225-229
9512, 225-229
9519, 228
AMDASM, 176
Analog interfacing, 230-236
Analog microprocessor, 235-236
Arithmetic processor, *see* Floating point processors
Arrhenius equation, 256
Assembly language, 115

Balance of trade, 19
Bit-slice microprocessors, 4, 148-166
Bit stuffing/stripping, 223
Blanking circuit, 213
Block replicate, 200
Blooming, 207
Boolean expression evaluation, 60-61
Bucket brigade device, 204-206
Buffering, 191
Bulk channel CCD, *see* Peristaltic CCD
Bus timing (**8086**), 83-84

Cassette memory, 184
CCD, *see* Charge coupled device
CCD memory organization, 207-208
CDI, *see* Collector diffusion isolation
Charge coupled device (CCD), 204-208
Charge transfer device, 204-208
Chevrons, 199
Circuit design, 14
Clean room, 198
Code segment, 76
Collector diffusion isolation (CDI), 63
Commodity stage, 6
Components, 241
Computer control unit, 170
Conductor access, *see* Current access
Continguous disks, 199
Core memory, 184
Cost/gate, 7
CRC, 193
Cross talk, 263
CRT interfacing, 213-220
Current access, 199

D/A converter, 230, 234-235
DAS, 166
Data General:
MBC/1, 63
mN 602, 63
MP/100, 63
MP/200, 64
Nova 4, 150
Data segment, 76
DDCMP, 223
DES, 225
Design-in, 6
Destination control, 153, 157
Development systems, 244-245
Device reliability, 255-263
Diagnostics, 148
Die size, 8, 10
Direct memory access (DMA), 209
Distributor stocking, 6

DLCC, 221-235
Drag business, 4
Dry etching, 12
Dual slope, 233
Dummy detector, 200
Dynamic life test, 257
Dynamic RAM, 184

EBCDIC, 183
ECL, 64, 158
Electron beam, 11
Encrypted data link interfacing, 225
Error checking, 166
Error correcting, 166
Evaluation systems, 241-244
Evolution, technological, 5-6
Exports, 18
External error detection, 264-265
Extra segment, 76

Failure mechanism, 255-257
Fairchild:
9440, 51-55
9445, 54
100220, 162, 166
Ferranti **F-100L**, 63
Field access, 199
Field inversion, 89
Field test equipment, 253-254
Figure of merit, 6, 8-9
Fixed disk, 197
Flag register, 75
Floating point processors, 225-229
Floppy disks, 193-197
Frame character detection, 223
Frequency modulation, 193

Gallium arsenide, 13
General registers, 74-75
Generations, microprocessor, 7

HDLC, 223
Hexadecimal representation, 114, 185
HMOS, 8
Hybrid circuit, 235

IBM:
3310, 198
3348, 198
3740, 194-195
Imports, 18
Instruction set:
8086, 82-114
Z8000, 134-147
Intel:
1101, 185
1103, 185
2102, 186
2107, 185
2125, 186
2920, 235-236
7110, 203-204
8080, 74-76
8086, 34-35, 77-114
8088, 67
8205, 210-212
8255, 210-213
8288, 79-80
Interfacing:
mechanical, 182
physical, 182
RAM, 191-192
Interrupt:
8086, 82
Z8000, 128, 133
nonmaskable, 82, 128
nonvectored, 128
vectored, 128
I/O interface control, 209-213

Japanese challenge, 27-31
Josephson circuits, 13

Label, 115
Laser annealing, 12
Latch-up, 262
Leadership, 21
Lift-off, 12
Limited precision, 61
Line width, 12
Lithography, 11
Logic analyzers, 246-253
Low-end microprocessors, 3
LSI-11 bus, 238-240
LTPD, 259

Magnetic bubbles, 198-204
Magnetic tape, 184
Magnetoresistive effect, 200
Major loop/minor loop, 199
Market segmentation, 2
Market share, 15
Mass memories, 192-208
Maturity of technology, 183
Maximum mode, 76-78
Memory mapped I/O, 209
Microcode, 152
Microinstructions, 149, 167
Microprocessor generations, 7
Microprogramming, 148-149
Microprogram sequencer, 150
Midrange enhanced microprocessors, 3
Minimum mode, 76-77
Mostek **4116**, 185
Motorola:
6809, 67
10800, 158, 160-162
10801, 158, 163
10802, 158, 164
10803, 158, 165

68000, 57-63
MSU, 166
Multibus, 237-238
Multiprotocol data link, 223-225

National Semiconductor:
NS 16000, 64
System/400, 150
Noise, 263-264
Nonvolatile RAM, 189
Nyquist's theorem, 235

Obsolescence, technological, 6
Offshore assembly, 17
One-transistor memory cell, 186-188

Pascal (program), 62
Patents, 22-27
PDA, 259
Performance trends, 8
Peristaltic CCD, 206
Permalloy, 199
Pertec **D8000**, 198
Photomasking, 11
Piccolo (disk), 198
PIU, 166
Plasma deposition, 12-13
Precharge circuit, 186
Privileged instruction, 129
Procedure calls, 61
Productivity, 20
Programmed I/O, 209
Program status area (**Z8000**), 31, 130
Pro-log System Analyzer, 253-254

Quality assurance, 259-261

Random failure, 256
Redundant systems, 264
Reliability, 181, 255-265
Relocation, 129
Repairability, 183
Research and development, 21

SDK-86, 242-244
Second sourcing, 6
Segmentation:
memory (**8086**), 82
trap, 129
Segment registers, 76
Self-diagnosis, 264
Semiconductor shipments, 15-16
Sensors, 231-232
Serial access, 183
Single board systems, 271-281
Sputter etching, 12
Stack segment, 76
Standard Microsystems **5027**, 214-220
Static RAM, 184
Status lines:
8086, 78
Z8000, 134
Storage capacity, 183
Successive approximation, 233
Superconductor technology, 13
Surface channel CCD, 206

Technological evolution, 5-6
Technological obsolescence, 6
Texas Instruments:
SN 54S/74S-481, 162
TMS 9900, 37-43
TMS 9940, 44-51
Three-transistor cell, 185-187
Timing diagram, **8086**, 79
Transducer, 230
Transparency, 223
Traps, 58, 128-130

UART, 220
Unimplemented instruction, 128-129
USART, 220
USASCII, 183
USRT, 220

Volatility, 183
VTAC, 215

Wafer fabrication, 11
Wearout, 256-257
Winchester disk, 198

X.25, 223
X-Ray photomasking technique, 11

Yen, 27-28
Yield, 10

Zilog:
Z80, 131
Z8000, 35-37, 127-147
Z8001, 127-128
Z8002, 127-128
Z8010, 130, 134